

UNIVERSIDAD DE COSTA RICA
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE0499 – Proyecto Eléctrico

**Teleoperación de un robot humanoide NAO utilizando un
sistema de captura de movimiento**

por

Lennon Núñez Meoño

Ciudad Universitaria Rodrigo Facio

Diciembre de 2017

Teleoperación de un robot humanoide NAO utilizando un sistema de captura de movimiento

por

Lennon Núñez Meoño

B34943

IE0499 – Proyecto Eléctrico

Aprobado por

Dr.rer.nat Francisco Siles Canales

Profesor guía

Elvira Chaves Pochet, Licda.

Profesor lector

Ricardo Román Brenes, M.Sc.

Profesor lector

Diciembre de 2017

Resumen

Teleoperación de un robot humanoide NAO utilizando un sistema de captura de movimiento

por

Lennon Núñez Meoño

Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Profesor guía: Dr.rer.nat Francisco Siles Canales

Diciembre de 2017

Los robots humanoides son vistos con más frecuencia como próxima herramienta común para ayudar en las tareas cotidianas, por ello es importante la inversión en el desarrollo de aplicaciones que permitan un mejor y más sencillo manejo de sus capacidades. Un método de control recurrente al utilizar la mayoría de los robots es la teleoperación, que dota de nuevas capacidades remotas al operador humano, permitiéndole controlar el robot desde un lugar seguro y cómodo.

Dada la similitud entre la anatomía humana y la del robot humanoide, un sistema que aprovecha esta condición en un control intuitivo es aquel que hace uso de la captura de movimiento, que permite rastrear el movimiento de diferentes marcadores colocados en el cuerpo del operador, con lo que solo se requeriría que la persona se mueva con naturalidad y a como espera que el robot lo haga.

El presente proyecto consiste en el desarrollo de la primera versión en el PRIS-Lab de un sistema de teleoperación con el cual se logre que un robot humanoide NAO opere como imitador de movimientos humanos, usando un Sistema de captura óptica de movimiento.

Palabras claves: *Captura, control, humanoide, imitador, marcador, movimiento, NAO, óptica, rastreo, remoto, robot, teleoperación.*

Acerca de IE0499 – Proyecto Eléctrico

El Proyecto Eléctrico es un curso semestral bajo la modalidad de trabajo individual supervisado, con el propósito de aplicar estrategias de diseño y análisis a un problema de temática abierta de la ingeniería eléctrica. Es un requisito de graduación para el grado de bachiller en Ingeniería Eléctrica de la Universidad de Costa Rica.

Abstract

Teleoperación de un robot humanoide NAO utilizando un sistema de captura de movimiento

Original in Spanish. Translated as: “Teleoperation of a NAO humanoid robot using an Optical Motion Capture System”

by

Lennon Núñez Meoño

University of Costa Rica

Department of Electrical Engineering

Tutor: Dr.rer.nat Francisco Siles Canales

December of 2017

The humanoid robots are being seen more frequently as common tools to aid the humans on daily life tasks, that is why it is important to invest into developing applications that allow a better and easier way of using its capacities. A common control method used in most of the robots is the teleoperation, which gives new remote capacities to the human operator, allowing the person to operate the robot from a safe and comfortable place.

Given the resemblance within the human anatomy and that of the humanoid robot, a very intuitive control system for the human that takes advantage of that similarity is that of motion capture, which tracks the motion of different markers placed along the body of the operator, then, all what it takes is to move naturally as the person expects the robot to do.

This project consist of the development of the first PRIS-Lab’s teleoperation system testing version intended to allow the NAO humanoid robot to perform as a human motion imitator, using an optical motion capture system.

Keywords: *Capture, control, humanoid, imitator, motion, NAO, optical, remote, robot, track, teleoperation.*

About IE0499 – Proyecto Eléctrico (“Electrical Project”)

The “Electrical Project” is a course of supervised individual work of one semester, with the purpose of applying design and analysis strategies to a problem in an open topic in electrical engineering. It is a requisite of graduation for the Bachelor of Science in Electrical Engineering, granted by the University of Costa Rica.

Dedicado a mi familia.

Agradecimientos

A mi familia por el apoyo que siempre me han dado y el esfuerzo de mis padres por darme siempre una buena educación, por todos sus sacrificios.

A mí mismo por no desaprovechar estas oportunidades y hacer algo con ellas.

A Elvira Chaves, Sofía Fonseca y Ricardo Soro por sacar de su tiempo para ayudarme con pruebas y grabaciones para que este proyecto pudiera avanzar.

A Raúl Lizano por la ayuda con diferentes tomas de fotografía y video para recopilar evidencia del desarrollo del proyecto.

A Dios por permitirme estas experiencias.

Índice general

Índice general	xii
Índice de figuras	xiv
Nomenclatura	xxiii
1 Introducción	1
1.1. Alcance del Proyecto	2
1.2. Justificación y Antecedentes	3
1.3. Trabajos similares consultados	5
1.4. Objetivos	6
1.4.1. Objetivo General	6
1.4.2. Objetivos Específicos	6
1.5. Metodología	8
1.5.1. Conocer las herramientas de trabajo	9
1.5.2. Desarrollo de una plataforma para la adquisición y procesamiento de datos de ubicación espacial	9
1.5.3. Implementar un sistema de teleoperación para el control de movimientos sencillos del robot humanoide NAO	9
1.5.4. Validar el funcionamiento del sistema de teleoperación del NAO	10
1.5.5. Documentar y publicar la implementación y resultados del proyecto	10
2 Teoría del trabajo escrito	11
2.1. Teleoperación y telepresencia	11
2.1.1. Similitudes y diferencias	11
2.1.2. Sistema de teleoperación básico	12
2.2. Herramientas de trabajo	14
2.2.1. Sistema de captura óptica de movimiento	14
2.2.2. Robot Humanoide NAO	23
2.3. Ajuste Humano - NAO y modelos para la captura de movimiento	31
2.3.1. Comparación del cuerpo humano con el del robot humanoide NAO	36

2.3.2. Regresión lineal: Generalidades	38
2.3.3. Diseño de un proceso de calibración	42
3 Implementación	45
3.1. Operaciones necesarias que debe cumplir el sistema de teleoperación	46
3.2. Lectura y escritura de archivos CSV	48
3.3. Obtención y generación de datos de posición espacial	52
3.4. Calibración del sistema de teleoperación	52
3.5. Ajuste de datos de una grabación del MoCap	53
3.6. Movimiento del robot humanoide NAO imitando la grabación	55
3.7. Uso del sistema de teleoperación paso a paso	59
3.7.1. Preparación del ambiente de trabajo	59
3.7.2. Generación de los datos de calibración del NAO	62
3.7.3. Calibrar el sistema de teleoperación	64
3.7.4. Teleoperar el robot humanoide NAO	64
4 Validación del sistema de teleoperación	67
4.1. Confirmación de la intención del movimiento	68
4.2. Evaluación de los datos usados por el sistema de teleoperación	68
5 Conclusiones y recomendaciones	73
5.1. Conclusiones	73
5.2. Recomendaciones y lecciones aprendidas	74
5.2.1. Consideraciones sobre el sistema de teleoperación desarrollado	75
5.2.2. Propuesta de mejoras del sistema de teleoperación	76
A Resultados completos de la validación del sistema de teleoperación	77
A.1. Intención de movimiento	77
A.2. Comparación gráfica del ajuste de los datos	77
A.2.1. Polinomio grado 1	78
A.2.2. Polinomio grado 2	93
B Proceso de Calibración	109
B.1. Calibración de los Brazos (RArm y LArm)	111
B.2. Tórax	113
B.3. Pierna Izquierda	114
B.4. Pierna Derecha	115
C Código del sistema de teleoperación	117
C.1. ErrorFunc.py	117
C.2. Calibración	119
C.2.1. Calibrate.py	119
C.2.2. Calibratefunc.py	123

C.2.3.	OffsetFileFunc.py	132
C.3.	Obtención datos del NAO	134
C.3.1.	GetPositions.py	134
C.4.	Ejecución de la teleoperación	139
C.4.1.	Move.py	139
C.4.2.	CSVMOCAPFunc.py	147

Índice de figuras

1.1.	Diagrama con 4 fases generales para teleoperar el robot humanoide NAO	8
2.1.	Generalidad de un modelo de un sistema de teleoperación. Requiere de un operador (persona), un sistema de control (diseño depende de la aplicación) y de un sistema operado (robot NAO)	13
2.2.	MoCap en las instalaciones del PRIS-Lab, área con solo 8 cámaras, tomado en Noviembre 2017	14
2.3.	Cámara de la serie Prime 41.	15
2.4.	Configuración física del MoCap en el PRIS-Lab. [45]	15
2.5.	Plano del espacio físico disponible del MoCap instalado en el PRIS-Lab. [45]	16
2.6.	Traje y marcadores para realizar grabaciones por MoCap	17
(a).	Traje especial para las grabaciones en MoCap, con algunos marcadores colocados	17
(b).	Marcadores del MoCap, grande y pequeño	17
2.7.	Preparación del robot NAO antes de iniciar una grabación con el MoCap	18
(a).	Marcador del MoCap colocado sobre pie del robot NAO, sostenido por cinta adhesiva	18
(b).	Robot NAO con emisores LED de los ojos y cabeza cubiertos para que no interfieran en las tomas	18
2.8.	Vara y escuadra para calibrar el MoCap	19
(a).	Vara de calibración	19
(b).	Escuadra para definir ejes del origen en el plano de referencia	19
2.9.	Modelo de cuerpo completo de 49 marcadores visto en Motive.	20
2.10.	Modelo de cuerpo completo de 42 marcadores en Motive.	22
2.11.	Configuración Física del Robot Humanoide NAO modelo H25.	23
2.12.	Posiciones predefinidas recomendadas.	25
(a).	Stand	25
(b).	Crouch	25
2.13.	Actuadores y Cadenas del robot humanoide NAO.	26
(a).	Actuadores y sus grados de libertad. Extraído de: http://doc.aldebaran.com/2-1/family/nao_dcm/actuator_sensor_names.html	26
(b).	Cadenas de actuadores. Extraído de: http://doc.aldebaran.com/2-1/family/robots/bodyparts.html	26
2.14.	Marcos de referencia TORSO y ROBOT.	27

2.15. Comparación entre los cuerpos de una persona y un robot humanoide NAO H25	32
(a). Modelo de 6 puntos que relaciona los actuadores finales de las cadenas de acción del NAO con sus equivalentes en puntos de control del cuerpo humano	32
(b). Comparación entre tamaños del cuerpo de una persona (1,7 m de altura) y un robot NAO (0,573 m de altura)	32
2.16. Ejemplo de múltiples soluciones para una misma ubicación espacial del actuador final de una cadena de acción	33
(a). Primera posible solución para la ubicación del actuador final	33
(b). Segunda posible solución para la ubicación del actuador final	33
(c). Ambas posiciones son soluciones viables si se considera solo la ubicación espacial (X,Y,Z)	33
(d). Al considerar rotación para la ubicación espacial, (X,Y,Z,wX,wY,wZ), ambas posiciones se vuelven diferenciables	33
2.17. Colocación de los marcadores del modelo con un cuerpo rígido por actuador final de las cadenas de acción del NAO, usando un total de 27 marcadores.	34
(a). Marcadores de la cabeza	34
(b). Marcadores de las manos	34
(c). Marcadores de los pies visto de frente	34
(d). Marcadores de los pies visto de lado	34
(e). Marcadores del torso visto por detrás	34
(f). Marcadores del torso visto de lado	34
2.18. Modelo de cuerpos rígidos con 27 marcadores totales visto desde Motive, sin marcadores perdidos.	36
(a). Ejemplo de cuerpo rígido del Torso (azul) con marcadores perdidos	36
(b). Modelo de cuerpos rígidos completo	36
2.19. Centros de masa de los 6 actuadores finales de las cadenas de actuadores a controlar del robot humanoide NAO, identificados como un marco de referencia de 3 ejes X, Y, Z.	37
(a). COM cabeza	37
(b). COM manos	37
(c). COM pies	37
(d). COM torso	37
2.20. Resultado del ajuste de grado 1 manual realizado a los datos del eje X para el movimiento de elevación y retorno a posición inicial de la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 marcadores.	38
2.21. Relación entre datos del NAO vs los del MoCap para la cadena RArm, sobre el eje X. Grabación de 10 s subiendo y bajando los brazos. Modelo de 6 marcadores. Eje X del MoCap apunta en dirección opuesta al del NAO.	39
2.22. Relación entre datos del NAO vs los del MoCap para la cadena RArm, sobre el eje Z. Grabación de 10 s subiendo y bajando los brazos. Modelo de 6 marcadores	40

2.23. Resultado del ajuste polinomial de grado 2 realizado a los datos del eje X para el movimiento de elevación y retorno a posición inicial de la cadena de actuadores RArm del robot humanoide NAO. Marco de referencia ROBOT. Modelo de 6 marcadores. Regresión por función <i>polyfit</i> de Python.	41
2.24. Resultado del ajuste de grado 2 realizado a los datos de coordenada Y para el movimiento de elevación y retorno a posición inicial de la cadena de actuadores LArm del robot humanoide NAO con datos provenientes de otra cadena. Marco de referencia ROBOT. Modelo de 6 marcadores. Regresión por función <i>polyfit</i> de Python.	42
3.1. Estructura del código del sistema de teleoperación	47
3.2. Ejemplo de visualización en un LibreOffice Calc de un archivo con datos de cuerpos rígidos exportado a CSV desde Motive	50
3.3. Barra de acciones de Choregraphe con ícono de <i>realizar conexión</i>	60
3.4. Interfaz Choregraphe con NAO conectado y rutina cargada	60
(a). Visualizador virtual del NAO conectado	60
(b). Bloque de rutina de calibración <i>Brazos</i> cargado en Choregraphe	60
3.5. Barra de acciones con ícono <i>wake up</i> (extremo derecho) y activación del modo animación (extremo izquierdo)	62
3.6. Elementos del archivo CSV de una persona correspondientes al actuador NAO eliminados y reemplazados por <i>Empty</i> como parte del proceso de calibración	64
4.1. Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	69
4.2. Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	69
4.3. Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	70
4.4. Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	70
4.5. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	71
4.6. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	71
A.1. Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	78
A.2. Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	78

A.3. Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	79
A.4. Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	79
A.5. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	80
A.6. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	80
A.7. Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	81
A.8. Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	81
A.9. Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	82
A.10. Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	82
A.11. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	83
A.12. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	83
A.13. Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	84
A.14. Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	84
A.15. Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	85
A.16. Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	85
A.17. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	86
A.18. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	86

A.19. Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	87
A.20. Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	87
A.21. Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	88
A.22. Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	88
A.23. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	89
A.24. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	89
A.25. Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	90
A.26. Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	90
A.27. Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	91
A.28. Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	91
A.29. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	92
A.30. Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	92
A.31. Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	93
A.32. Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	93
A.33. Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	94
A.34. Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	94
A.35. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	95

A.36. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	95
A.37. Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	96
A.38. Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	96
A.39. Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	97
A.40. Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	97
A.41. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	98
A.42. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	98
A.43. Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	99
A.44. Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	99
A.45. Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	100
A.46. Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	100
A.47. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	101
A.48. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	101
A.49. Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	102
A.50. Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	102
A.51. Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	103
A.52. Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	103

A.53. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	104
A.54. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	104
A.55. Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores Tórso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	105
A.56. Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores Tórso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	105
A.57. Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores Tórso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	106
A.58. Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores Tórso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	106
A.59. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores Tórso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	107
A.60. Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores Tórso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.	107
B.1. Pose Stand, inicial y final para todas las rutinas de calibración	110
(a). Posición de la persona equivalente al Stand del robot humanoide NAO.	110
(b). Posición Stand del robot humanoide NAO.	110
B.2. Calibración de brazos. Paso 1, NAO.	111
(a).	111
(b).	111
B.3. Calibración de brazos. Paso 1, persona.	111
B.4. Calibración de brazos. Pasos 2 y 3, NAO.	112
(a). Calibración brazos - Paso 2	112
(b). Calibración brazos - Paso 3	112
B.5. Calibración de brazos. Pasos 2 y 3, persona.	112
(a). Calibración brazos - Paso 2	112
(b). Calibración brazos - Paso 3	112
B.6. Calibración del Tórso. Pasos 1 y 2, NAO.	113
(a). Calibración torso - Paso 1	113
(b). Calibración torso - Paso 2	113
B.7. Calibración del Tórso. Pasos 1 y 2, persona.	113
(a). Calibración torso - Paso 1	113
(b). Calibración torso - Paso 2	113
B.8. Pierna Izquierda. Pasos 1 y 2, NAO.	114

(a). Calibración pierna izquierda - Paso 1	114
(b). Calibración pierna izquierda - Paso 2	114
B.9. Pierna Izquierda. Pasos 1 y 2, persona.	114
(a). Calibración pierna izquierda - Paso 1	114
(b). Calibración pierna izquierda - Paso 2	114
B.10. Pierna Derecha. Pasos 1 y 2, NAO.	115
(a). Calibración pierna derecha - Paso 1	115
(b). Calibración pierna derecha - Paso 2	115
B.11. Pierna Derecha. Pasos 1 y 2, persona.	115
(a). Calibración pierna derecha - Paso 1	115
(b). Calibración pierna derecha - Paso 2	115

Nomenclatura

Choregraphe Aplicación de escritorio multi-plataforma que permite el control y monitoreo del robot humanoide, así como la creación de animaciones y diálogos para el robot.

CORE División del PRIS-Lab cuyos esfuerzos se dedican a la exploración de la robótica, cubriendo diferentes áreas de la ciencia y el arte, promoviendo la interacción humano-robot y buscando la creación de sistemas inteligentes para permitir a robots el razonar sobre su entorno, resolviendo problemas complejos para interactuar mejor con éste.

MoCap (Sistema de) Captura de Movimiento, siglas en inglés *Motion Capture*.

Motive Plataforma de software diseñada por OptiTrack para el control de varios sistemas de captura de movimiento.

NAO Robot Humanoide autónomo y programable desarrollado por ALDEBARAN Robotics, lanzado al público en 2008.

OptiTrack Industria líder mundial como proveedor de sistemas de captura óptica de movimiento, incluyendo tanto el equipo de hardware como las plataformas de software.

PRIS – Lab Laboratorio de investigación en reconocimiento de patrones y sistemas inteligentes de la Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica, siglas en inglés *Pattern Recognition and Intelligent Systems Laboratory*.

ROS Ambiente de desarrollo de software conformado por una colección de herramientas, librerías y convenciones con el objetivo de simplificar la tarea de crear software complejo y robusto para la robótica, del inglés *Robot Operating System*

SDK Set de herramientas de desarrollo de software, siglas en inglés *Software Development Kit*.

ALDEBARAN Robotics Empresa de robótica francesa, creadores del robot humanoide Nao. Adquirida por SoftBank Group en 2015.

API Set de subrutinas, protocolos y herramientas para la construcción de una aplicación de software, siglas en inglés *Application Programming Interface*

BVH Formato de datos de captura de movimiento desarrollado por Biovision que presenta la información como una jerarquía esquelética del cuerpo capturado, del inglés *Biovision Hierarchy*

- COM Centro de Masa, siglas en inglés *Center of Mass*
- CSV Formato de archivo cuyo contenido está escrito en *texto plano* y es interpretado como datos organizados tabularmente, separados por coma, siglas del inglés *Comma-Separated Values*
- DoF Grados de Libertad, siglas en inglés *Degrees of Freedom*
- EIE Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica.
- IEEE Instituto de Ingenieros Eléctricos y Electrónicos, del inglés *Institute of Electrical and Electronics Engineers*.
- SoftBank Group Corp. Corporación multinacional japonesa de telecomunicaciones e internet.
- SoftBank Robotics Corp. División de SoftBank Group Corp. entre los líderes mundiales en el campo de la robótica humanoide, anteriormente ALDEBARAN Robotics.
- Texto plano Texto que no está escrito en código o especialmente formateado, entiéndase del inglés *plain text*

Capítulo 1

Introducción

De la mano con el cambio de los ambientes laborales a través del tiempo ha estado presente el deseo por simplificar las tareas y esfuerzos que requiera el ser humano para cumplir con las mismas, desde el diseño y desarrollo de diferente maquinaria para ayudar en la ejecución de las actividades de trabajo hasta la idea de creación de mecanismos automatizados que lleguen a sustituir a las personas como fuerza laboral; tal como se plantea en distintas obras literarias como *R.U.R.*¹ del autor checo Karel Čapek [17], obra que da a conocer al público la palabra Robot² y la relación de estos robots como ayuda para el ser humano, o bien, las obras de Isaac Asimov [24], como *Círculo Vicioso* y el conjunto de obras de la *Serie de la Fundación*, donde se aborda de manera extensa ciertas implicaciones de la interacción de los humanos con los robots y la evolución de los mismos.

La robótica es ampliamente aplicada en la industria al automatizar diferentes procesos, de modo similar se trata de involucrar a estos robots en las tareas domésticas para ayudar al ser humano en su cotidianidad. Esto con diseños adaptados a los medios domésticos [23], incluyendo los robots con anatomía semejante a la humana (humanoide). Dada su creciente influencia en las actividades diarias y el potencial de aprovechamiento de sus funciones, en especial para los robots humanoides al tener mayor facilidad de adaptarse a los medios urbanos creados para las personas por la similitud entre estas anatomías, es importante el desarrollo de diferentes medios de control para estos robots, de modo que estos puedan efectuar las tareas solicitadas tal cual se quiere que las realicen, manteniendo la comodidad y seguridad para el humano que le opera.

Dentro de los mecanismos para utilizar estos sistemas electromecánicos existe el concepto de tele-robótica [48], donde se ejerce control a distancia sobre el robot, en la actualidad explotando las facilidades que ofrecen las conexiones inalámbricas. Esto involucra dos campos de operación o modo de acción, por un lado la generalidad del control a distancia sobre el sistema de interés o teleoperación, y por otro lado, la relación del sistema y la capacidad del ser humano para sentir la presencia de los otros a través de la interacción con el sistema bajo operación o telepresencia.

Así como existen diferentes enfoques para la operación del robot, es posible ejercer el control con diversas herramientas, esto según el área de aplicación. El presente proyecto hace uso de las facilidades del sistema de captura óptica de movimiento (MoCap) en posesión del PRIS-Lab para ejercer la

¹Robots Universales Rossum, del checo *Rossumovi univerzální roboti*

²Del checo *robota*, que quiere decir trabajo [29]

teleoperación de un robot humanoide NAO, haciendo del robot un imitador de movimientos humanos [28, 44, 52] a través de una primera versión de la plataforma del control teleoperado, con énfasis en el movimiento de los brazos.

El presente trabajo está compuesto por 6 capítulos, los cuales están concebidos de la siguiente manera: el presente corresponde al capítulo 1, cuyo contenido incluye la introducción, donde se abarca los alcances y planteamiento del proyecto, resultado de investigación y experimentación; el capítulo 2 trata la teoría y generalidades técnicas necesarias para el desarrollo del proyecto; el capítulo 3 hace una breve descripción de la implementación de la base operacional del sistema de teleoperación; el capítulo 4 incluye el proceso seleccionado para la validación del sistema y algunos resultados obtenidos, así como problemas encontrados y cómo se afrontaron; el capítulo 5 contiene las conclusiones y resultados.

1.1. Alcance del Proyecto

El proyecto se enfoca en generar un compendio de información (el trabajo presente) cuyo contenido explique el proceso que abarca diseñar e implementar un sistema de software que permita efectuar de manera sencilla la teleoperación de un robot humanoide NAO a partir del movimiento del cuerpo de una persona. Este sistema pretende ser una base operacional para la teleoperación del robot humanoide por un sistema de captura de movimiento, por medio del cual se pueda emprender proyectos futuros de perfeccionamiento en su funcionalidad y de incorporación de nuevas acciones de control, como lo son un estudio detallado del balance del robot NAO partiendo de los requerimientos y limitaciones descritas en este proyecto, efectuar la teleoperación en *tiempo real*, la implementación de un sistema de calibración refinado para el control remoto del robot NAO, el refinamiento del movimiento del NAO al imitar el de una persona, ejecución de movimientos más complejos por el NAO y el desarrollo de diferentes modelos 3D que faciliten la captura y acople humano-NAO, todo esto partiendo del supuesto que para poder teleoperar correctamente al robot humanoide NAO primero se debe tener una base de teleoperación y por tanto lo principal para este proyecto es el lograr convertir datos del movimiento de una persona en instrucciones de ejecución del movimiento para el robot humanoide NAO.

Para el contexto del trabajo presente se entiende por teleoperación del robot humanoide NAO como la operación remota del robot a través de una red inalámbrica, de modo que este opere imitando el movimiento previamente capturado de un humano. La prioridad de imitación es de la colocación de los brazos, aunque se realizan capturas y pruebas con el movimiento de las piernas y torso en menor porción para dejar lista una base de coreografías cortas, pruebas y teoría que permita entrar con mayor facilidad al tema del balance y movimiento complejo del robot humanoide NAO en proyectos futuros de continuación y mejora del sistema de teleoperación. Entiéndase además **sistema como base operacional** como el hecho de que el proyecto consiste en un primer acercamiento de la creación de una plataforma en software para este control del robot y por ende no se pretende generar una versión que satisfaga todo movimiento ejecutable por una persona y capturable por el MoCap para que el NAO le ejecute, esto por el tiempo disponible para el desarrollo del proyecto en comparación con el tiempo de investigación y pruebas necesario para ejecutar movimientos complejos y balanceados por el NAO. Este proyecto define limitaciones en primera instancia y propone requerimientos para una próxima versión más completa y con mayores capacidades.

El diseño de la plataforma incluye pasos necesarios para llegar desde la obtención de los datos bases para el control hasta la ejecución de las tareas deseadas en el robot humanoide. Para la implementación se utiliza como sistema controlado el robot humanoide NAO con fisionomía tipo H25 de ALDEBARAN (detalles en el capítulo 2); como herramienta de obtención de datos se utiliza un sistema de captura óptica de movimiento conformado por el conjunto de cámaras de rastreo óptico de la serie *Prime 41* instalado dentro de las facilidades del PRIS-Lab, en la Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica, y la plataforma Motive para la recolección y visualización primaria de los datos obtenidos por las cámaras, ambas herramientas de OptiTrack; para el desarrollo del sistema de teleoperación se hace uso de las estructuras y métodos de los diferentes API's oficiales disponibles del robot humanoide NAO.

La teleoperación puede ser efectuada a partir de datos del movimiento humano obtenidos de dos maneras principales: la primera consiste en utilizar una serie de datos previamente almacenados (grabación) y obtenidos del rastreo del movimiento humano; en el segundo los datos del rastreo son obtenidos en *tiempo real* a través de la transmisión de los datos por una red inalámbrica usando Motive y el protocolo NatNet. El proyecto utiliza grabaciones editadas en Motive como método de obtención de datos, con el procesamiento de archivos exportados en formato CSV.

1.2. Justificación y Antecedentes

Dado que la anatomía de un robot humanoide es una derivación del cuerpo humano, es posible tener mayor claridad y relación directa con el movimiento que el robot sería capaz de realizar, si se ve desde lo acostumbrado que está el ser humano a controlar su propio cuerpo y entender su anatomía, sin embargo, las interfaces utilizadas para su operación no suelen ser intuitivas o fáciles de adaptar a las personas que desean ejercer control sobre el robot, ni suelen aprovechar esta semejanza física entre operador y operado. El proyecto toma ventaja de esta similitud entre la estructura del ser humano y una figura humanoide para generar una plataforma intuitiva entre el modo de acción y el resultado esperado, a través de un sistema de captura óptica de movimiento donde el usuario únicamente deberá mover su cuerpo a como espera que el robot lo haga.

Este método no solo disminuye el tiempo que se requeriría para entender cómo funciona la interfaz de control para obtener el resultado esperado, sino que facilita el proceso mismo de la operación del control del robot, ya que, en particular para el robot humanoide NAO, se suele recurrir al modo *Animación* que provee la plataforma *Choregraphe*. Este suele ser un método cómodo ya que permite la manipulación directa del robot humanoide como si se tratara de un títere, con gran similitud al proceso básico de la animación digital por cuadros en una línea temporal, sin embargo, requiere de invertir mucho tiempo en la definición y toma de las escenas que conformarán la animación, además de la necesidad de encontrar posiciones estáticas estables y que estas permitan un flujo balanceado entre las tomas, es decir, que el NAO no se caiga entre una pose y la próxima, todo de manera manual. También se simplificaría el proceso de la animación al solo tener que grabar con MoCap a una persona que efectúe los movimientos que se desean.

El proyecto es además un aporte a los objetivos de la división CORE del PRIS-Lab, uno de los cuales es participar en el evento internacional RoboCup [22], donde los robots humanoides NAO se enfrentan

en equipos en un campeonato de fútbol. En este campeonato los robots deben operar de manera autónoma, sin que un humano lo controle directamente y en tiempo real, de modo que es importante que los robots encuentren las mejores posiciones para desplazarse y patear el balón por sí solos partiendo de alguna base de datos en ellos. El proyecto pone una base que será utilizada en futuro para enseñar, por ejemplo, mejores maneras de patear el balón, por medio de la captura de movimiento de una persona ejecutando la acción para que el NAO imite la animación, esto apoyado por la interdisciplinariedad con otras divisiones del PRIS-Lab como ACE, que tiene proyectos de desarrollo de algoritmos de detección de patrones para extraer tácticas y estrategias de partidos de fútbol humano, y MOVE, con proyectos de análisis tridimensional del movimiento humano para mejorar métodos de terapia física y desempeño deportivo. Con esta conexión sería posible mejorar las capacidades individuales de juego por parte del robot.

Además de la motivación de participar en RoboCup y de la colaboración con las demás divisiones del PRIS-Lab, dentro de CORE se han desarrollado diversos proyectos para entender las diferentes funciones del robot, como el proyecto NAO IR para establecer comunicación por luz infrarroja entre dos robots, así como proyectos para mejorar distintas funcionalidades del NAO, como la implementación de un algoritmo de odometría visual monocular para mejorar las capacidades de ubicación y desplazamiento del robot humanoide NAO [51]. Estos diferentes proyectos permiten tener un mejor control sobre el robot humanoide NAO y con ello poder utilizarlo con mayor libertad y posibilidad de acción en diferentes actividades como el juego de Enano-Gigante por medio de control remoto por joystick, o la participación del robot en coreografías de danza y obras de teatro.

Dentro del área de teleoperación existen el proyecto NAO Remote, enfocado en el control remoto del robot por medio de un joystick, y otros más en proceso que incluyen al robot dentro de las artes escénicas. Cabe mencionar también el proyecto Puppeteer, en desarrollo por Pablo Ávila, que teleopera el robot por medio del dispositivo Leap, además de abrir las puertas a proyectos en telepresencia al involucrar el uso de visores de realidad virtual; y el proyecto en desarrollo por Eliécer Abarca para controlar el NAO con EEG³, ambos proyectos con aplicación enfocada en personas con discapacidades motoras.

También se debe mencionar proyectos como el de Mario Castresana [18], que proporciona de un algoritmo para la corrección de glitches en grabaciones realizadas por el MoCap, con gran aporte al futuro del uso del proyecto presente dado que se utilizan grabaciones por MoCap que pueden incluir de estos fallos en los datos obtenidos y reflejarlos directamente en el movimiento final del robot NAO.

En conjunto a estos y otros proyectos por venir el trabajo actual dota de nuevas capacidades de entendimiento sobre el funcionamiento del robot humanoide NAO, en especial para el control de cuerpo completo y balance, y da una nueva herramienta para su operación, abriendo las puertas a una nueva gama de actividades partiendo de la captura óptica del movimiento humano, por ejemplo, para facilitar la integración del NAO en obras de teatro y danzas.

³Electroencefalografía, método de monitoreo de la actividad cerebral por señales eléctricas, del inglés *Electroencephalography*.

1.3. Trabajos similares consultados

La teleoperación puede efectuarse siguiendo diferentes caminos, la selección del camino que guía el desarrollo del sistema de teleoperación presente no fue sencilla en especial por haber encontrado muchos caminos posibles y su contraste con el tiempo disponible, estas referencias consultadas permitieron sentar las bases de posibilidades y algunos conceptos por los que preocuparse para poder desarrollar el sistema, así como para saber dónde encontrar las respuestas a diferentes problemas encontrados luego de realizar pruebas al sistema.

La estructura del presente trabajo escrito como una guía de los pasos iniciales al comenzar un proyecto similar surge de cómo está presente la información en estos y otros textos semejantes consultados, en que en la mayoría se omitía esta información, o no era detallada del todo, enfocados en métodos para mejorar el movimiento final del NAO y el procesamiento de los datos obtenidos para un sistema de teleoperación ya existente. En este caso el sistema de teleoperación no existía y se ocupaba esa información omitida para crear el primer sistema de este tipo en el PRIS-Lab.

Dentro de las herramientas más utilizadas para la captura de movimiento está el sensor Kinect [31], para el cuál han sido desarrollados varios módulos dentro de ROS que permiten usar el Kinect para teleoperar robots humanoides [47]. Sin embargo, varios proyectos como [27, 52], utilizan sistemas de captura de movimiento como el Xsens MVN⁴, que se asemeja al utilizado en este proyecto en cuanto al uso de un traje especial con puntos identificados ya rastreados, proyectados a un espacio 3D virtual.

En cuanto al modo de operación, [26] decide realizar un paso de optimización de los datos y ajuste considerando las diferencias fisionómicas existentes entre el cuerpo humano del operador y del cuerpo humanoide del robot en operación previo a la ejecución del movimiento, con enfoque en el movimiento del área superior del cuerpo (brazos y torso), utilizando los pies para el balance; casos como [1, 27, 28, 52] realizan la teleoperación en *tiempo real*, con movimiento de cuerpo completo del robot humanoide NAO, con el extra de lograr *movimientos complejos* como levantar los pies sin que el NAO se caiga.

El modelo utilizado para relacionar los puntos de captura de la persona con los puntos de acción del NAO también varía bastante entre los proyectos. Algunos como [28] deciden efectuar control sobre 24 DoF del NAO por medio de una serie de matrices y transformaciones, calculando a partir de los datos obtenidos de la captura del Kinect los ángulos de ubicación de estos 24 DoF, usan coordenadas angulares en vez de rectangulares. Además propone el uso de poses personalizadas específicas para ser utilizadas como *transientes* entre posiciones en que el COM del NAO pueda ser apoyado sobre una pierna u otra, introduciendo entonces un sistema de balanceo flexible y dinámico. También se resalta la importancia de un sistema de balance durante la ejecución del movimiento, mostrando en un robot virtual cómo falla el realizar solo *mapeo directo* de las posiciones en el NAO al carecer de balance.

En cambio, [27] decide simplificar el modelo de relación de puntos de acción, separando las extremidades de los cuerpos de la persona y del robot como cadenas de acción, y relacionando estas con las del NAO para ejercer el control, con lo que logran poder enfocarse en los actuadores finales de las cadenas de acción del NAO y sus COM en vez de cada punto de acción disponible. Además propone un estudio sobre la proyección de la ubicación del COM del robot NAO para cambiar de manera dinámica

⁴Página web de XSens: <https://www.xsens.com/>

su apoyo según las restricciones de balance para que este pueda ser sobre la pierna izquierda, derecha o ambas según sea el caso.

Otros proyectos se enfocan en agregados al sistema de teleoperación al implementar diferentes visiones al control descrito en las referencias anteriores, como [52] al utilizar redes neuronales y todos los DoF disponibles del robot humanoide NAO para enseñarle a moverse como una persona, en contraste con el proceso de solo enviarle las posiciones directas a las que se quiere ubicar sus actuadores. Dentro del proceso se incluye una serie de rutinas de movimientos ejecutados por el NAO para que la persona a teleoperar los imite, siendo esta una calibración del sistema de la que se obtienen los datos para la comparación de movimientos del humano con los del robot. [25] menciona que las técnicas comunes para la teleoperación en tiempo real implican una gran carga computacional que impacta la latencia de la operación, y propone un modelo de proyección de los datos de las poses efectuadas en el espacio de configuración del humano en el espacio de configuración de un humanoide.

Estos trabajos permitieron la formulación de algunos requerimientos para el sistema diseñado, a pesar de que el presente proyecto no realiza la teleoperación en *tiempo real* como los trabajos consultados, o el uso de ubicación espacial por ángulos en vez de coordenadas rectangulares. También fue posible encontrar respuestas a problemas encontrados durante las pruebas realizadas gracias a los términos y conceptos tratados en estos trabajos, que guiaron en cómo buscar las respuestas en las diferentes documentaciones oficiales disponibles para el control del movimiento del NAO, así como en diferentes foros sobre el desarrollo con el robot humanoide NAO, como la página web de la comunidad de desarrolladores y entusiastas en la robótica con softBank, <https://community.ald.softbankrobotics.com/>. Al consultar este ambiente se encontró que varios problemas encontrados en el desarrollo del sistema de teleoperación eran comunes dentro de la comunidad, como el balance del NAO y la existencia del administrador de balance, esto también motivó a que el trabajo presente tenga forma de guía en las herramientas disponibles para crear un sistema de teleoperación para responder a estos problemas concurrentes.

1.4. Objetivos

1.4.1. Objetivo General

Teleoperar un robot humanoide NAO utilizando un sistema de captura óptica de movimiento.

1.4.2. Objetivos Específicos

- Investigar sobre la teleoperación, las herramientas disponibles para su funcionalidad y su relación con el control de cuerpo completo del robot humanoide NAO, y los sistemas de captura óptica de movimiento.
- Diseñar un sistema en software que permita la teleoperación de un robot humanoide NAO por captura de movimiento.
- Implementar la primer versión del PRIS-Lab de un sistema para la teleoperación de un robot humanoide NAO por captura óptica del movimiento de una persona.

- Validar el funcionamiento del sistema de teleoperación del robot humanoide NAO por captura óptica de movimiento a través de un plan de pruebas enfocado en la intención del movimiento.
- Documentar y publicar la implementación y resultados del proyecto.

1.5. Metodología

El proyecto consta de cuatro fases principales (Ver Figura 1.1) que resumen el proceso de diseño e implementación de la plataforma, estas son: obtención de datos, ajuste de datos, organización de datos y ejecución del movimiento, que guían la operación y uso del sistema desarrollado.

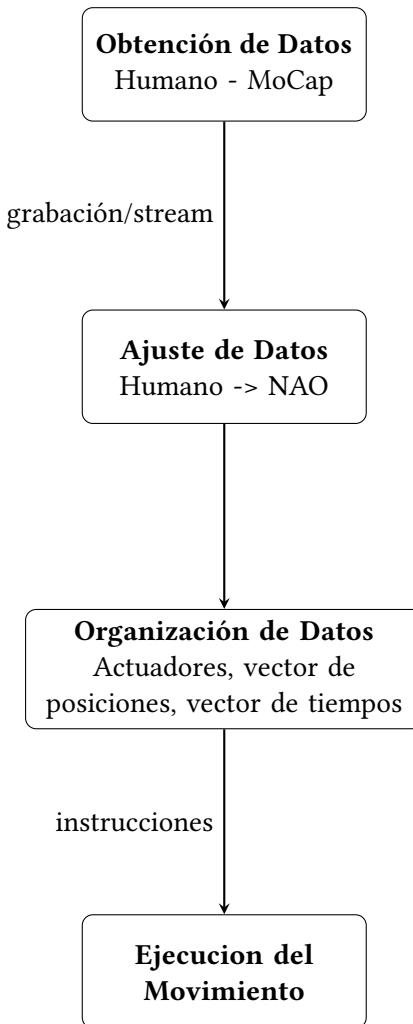


Figura 1.1: Diagrama con 4 fases generales para teleoperar el robot humanoide NAO

A continuación las tareas específicas que satisfacen el cumplimiento de los objetivos específicos.

1.5.1. Conocer las herramientas de trabajo

- Lectura de la guía de Usuario de Motive y de instalación de las cámaras Prime 41.
- Seguir los pasos de preparación del MoCap para una grabación.
- Realizar grabaciones de prueba con el MoCap y el NAO.
- Ejecutar scripts de ejemplos de ALDEBARAN del control de movimiento de cuerpo completo del NAO.
- Realizar captura de las posiciones de los actuadores del NAO por medio de las funciones intrínsecas de los API's del robot.
- Comprender las implicaciones de los distintos modos de ubicación espacial incluidos para el robot humanoide NAO, así como modelos para captura de datos.
- Estudiar el funcionamiento del balance automático en el robot humanoide NAO.

1.5.2. Desarrollo de una plataforma para la adquisición y procesamiento de datos de ubicación espacial

- Comprender los formatos de exportación de datos BVH y CSV, dando prioridad al CSV.
- Desarrollar un plan de adquisición de datos con posiciones para el NAO.
- Generar archivos CSV a partir de la captura de movimiento del NAO según el plan de adquisición de datos por medio de las funciones del SDK del robot humanoide NAO por lectura de sensores.
- Comprender el formato esperado por el NAO para ejecutar movimientos basados en coordenadas XYZ, luego XYZ con rotación.
- Desarrollar un módulo de programación que permita la lectura de un archivo CSV y su edición según el formato requerido para su procesamiento por el robot humanoide NAO.

1.5.3. Implementar un sistema de teleoperación para el control de movimientos sencillos del robot humanoide NAO

- Ejecutar scripts de ejemplo del balance de cuerpo completo automático del NAO y de movimiento de cuerpo completo.
- Generar archivos CSV a partir de la captura de movimiento de una persona según el plan de adquisición de datos.

- Comparar analíticamente los datos del rastreo por MoCap de una persona con los obtenidos de la lectura de los sensores del robot humanoide NAO obtenidos previamente.
- Obtener una relación matemática entre los datos de la persona y los del NAO para cada punto de acción (actuadores del NAO bajo control).
- Realizar ajustes en los datos de la persona según la relación matemática descrita con respecto a los datos del NAO en la plataforma de adquisición y procesamiento de datos.
- Desarrollar un módulo de programación que permita la extracción de los datos procesados y su transmisión hacia el NAO para la ejecución del movimiento.
- Transmitir los datos del movimiento hacia el NAO por medio de la plataforma para la teleoperación.
- Realizar ajustes sobre la ejecución del movimiento en el NAO utilizando el balance de cuerpo completo del API de movimiento del robot.

1.5.4. Validar el funcionamiento del sistema de teleoperación del NAO

- Desarrollar un plan de pruebas para la comprobación del sistema.
- Comparar los datos de captura de movimiento del NAO por el sistema de teleoperación con los de captura de la persona grabada con la dirección e intención del movimiento como criterio de comparación.

1.5.5. Documentar y publicar la implementación y resultados del proyecto

- Redactar un documento explicativo del proceso llevado a cabo y los resultados principales del desarrollo del sistema de teleoperación.
- Generar evidencia visual (video grabaciones) de los resultados principales del funcionamiento del sistema de teleoperación implementado.

Capítulo 2

Teoría del trabajo escrito

Al iniciar con el proyecto la información útil para comenzar con el mismo se encontró esparcida y con algunos detalles sin mencionar. El propósito de este capítulo es mostrar la información recopilada por investigación y experiencia en las pruebas efectuadas durante el desarrollo del sistema de teleoperación presente en una sola fuente para facilitar el inicio de proyectos similares, mostrando el camino seleccionado con sus ventajas y desventajas, así como de fuentes externas con información más detallada y otras propuestas para desarrollar un sistema de teleoperación por captura de movimiento aplicado al robot humanoide NAO.

La información a continuación incluye descripciones técnicas de las herramientas físicas utilizadas (MoCap, robot NAO)¹, así como consideraciones sobre su uso que son de utilidad tener de antemano al comenzar un proyecto similar y al utilizar el sistema de teleoperación desarrollado, no solo para entender el funcionamiento sino para saber razones y cómo responder ante diferentes situaciones y problemas que puedan surgir.

Las herramientas de desarrollo en software (Choregraphe, Motive, Python SDK) se describen brevemente en el capítulo 3 enfocado en su instalación como partes del ambiente de trabajo, y su uso dentro del sistema de teleoperación.

2.1. Teleoperación y telepresencia

2.1.1. Similitudes y diferencias

Dependiendo del alcance deseado al operar un sistema robótico se puede hablar solamente de teleoperación, o dar un paso más a la telepresencia [49].

Un sistema de teleoperación es definido por Matjaž Mihelj y Janez Podobnik [32] como aquel sistema que permite al ser humano explorar y manipular objetos de manera remota. Para ello el operador humano (**maestro**) debe enviar las instrucciones de operación y el robot (**esclavo**) deberá responder ante tales comandos para dotar de esas nuevas capacidades remotas a la persona que le opera. Un ejem-

¹Se recomienda en todo momento acudir a la documentación oficial de cada herramienta para entrar en mayor detalle a su uso

Plano de este caso es el brazo mecánico teleoperado Canadarm² [21], este mecanismo era operado desde el interior del transbordador espacial por un astronauta calificado, quien recibía ayuda de imágenes en tiempo real de la ubicación del brazo robótico para manipular carga como equipo de mantenimiento, inclusive era usado para transportar astronautas hasta satélites a tratar.

Ahora bien, si a esa capacidad de acción remota sobre el medio a través de un robot se le añade el ámbito de respuesta sensorial hacia el humano operador y aquellos otros que interactúen con la herramienta, se acerca el concepto de telepresencia [33]. Este término no excluye el de teleoperación, sino que añade una extensión presencial remota al operador humano. En el caso del Canadarm, su enfoque en operación y resultados esperados no era hacer que el astronauta operador sintiera que estaba fuera del transbordador espacial o que interactuara con otros astronautas en el exterior a como si estuvieran a su lado, sino dotarle de una extensión de control remoto sin que tuviera que salir del transbordador para efectuar las tareas en persona; en contraste, la telepresencia se puede ejemplificar con el robot BEAM³ [53], cuyo propósito no es dotar de capacidades de manipulación remota (aunque no excluye el caso en que pueda llegar a hacerlo), sino permitir que el humano operador interactúe con otras personas de manera remota, incluyendo micrófonos, parlantes, cámara y pantalla para dar la *sensación* a los involucrados de que están uno al lado del otro sin importar las distancias físicas reales. Importa ver y escuchar a la otra persona para interatuar como si estuvieran juntos.

Ambos conceptos comparten el mismo motivo de aplicación, dotar de una extensión remota al operador del sistema. Desde realizar tareas complicadas y peligrosas estando en una zona segura y cómoda, hasta conectar personas y acciones sin importar la distancia que haya de por medio. Estas ventajas hacen atractivos los sistemas teleoperados, en especial dentro de la robótica, generando así una gran gama de herramientas para el control y de modelos para ser operados.

En cuanto al proyecto, se considerará como teleoperación básica al hecho de poder accionar el robot humanoide NAO partiendo de los datos capturados desde el MoCap. Para esto ha sido diseñada una estructura que considera un camino posible para transformar los datos del MoCap en instrucciones para el NAO. Aparte de la teleoperación más simple está el criterio de evaluación que calificará la capacidad de este sistema de teleoperación, y por medio del cual definir ya sea otros caminos para ejercer el control remoto, o bien mejoras a esta base. Dicho criterio involucra la intención del movimiento (dirección) y la ubicación espacial deseada.

2.1.2. Sistema de teleoperación básico

A partir de las definiciones previas es posible describir un modelo sencillo con las partes necesarias para conformar un sistema de teleoperación (ver Figura 2.1). Dichas secciones necesarias son: **operador**, por el enfoque del proyecto actual se hace uso de un operador humano; **sistema de control**, que a su vez se puede subdividir en diferentes módulos funcionales y herramientas, según los requerimientos y complejidad de las tareas a ejecutar; y el **sistema operado**, en este caso entendido por el robot humanoide NAO.

²Conocido como SRMS, por sus siglas en inglés *Shuttle Remote Manipulator System*, desarrollado bajo la dirección del Consejo Nacional de Investigación de Canadá [19]

³Desarrollado por Suitable Technologies

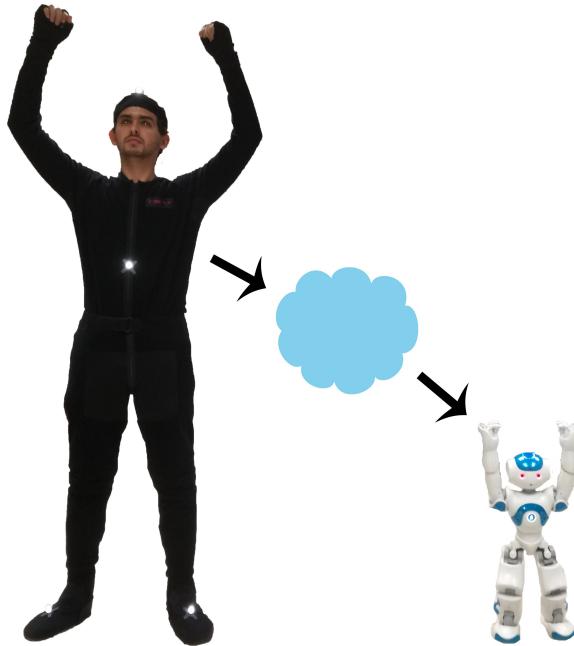


Figura 2.1: Generalidad de un modelo de un sistema de teleoperación. Requiere de un operador (persona), un sistema de control (diseño depende de la aplicación) y de un sistema operado (robot NAO)

Operador

Entiéndase como el individuo, sujeto humano, cuya intención es teleoperar el robot humanoide NAO. Dado el modo en que se usará el sistema de captura de movimiento, este individuo debe ser capaz de mover ambos brazos y piernas (en especial brazos). Esta primera versión del sistema de teleoperación se enfoca en el movimiento de brazos y torso, sin embargo, es importante considerar los efectos del movimiento de las piernas y el torso para el balance, estos conforman las principales causas de desbalance en el NAO.

Sistema de control

Conjunto de módulos operacionales encargados de cumplir con los tres primeros bloques definidos en el diagrama de la Figura 1.1, los cuales son: obtención de datos, ajuste de datos y organización de datos, inclusive el envío de instrucciones hacia el sistema controlado. Su entrada es el conjunto de datos del rastreo del movimiento de la persona (*operador*), y su salida son las instrucciones de control del movimiento del robot humanoide NAO (*sistema controlado u operado*).

Para la obtención de datos de captura del movimiento humano se utiliza un conjunto de cámaras de la Serie *Prime 41* (ver Figura 2.3) que conforman un sistema de captura óptica de movimiento, los datos que estos objetos proporcionan son obtenidos en primera instancia por el software Motive, desde

el cual se realiza las tomas requeridas y se exporta la información en un formato editable (CSV, BVH, entre otros).

El ajuste y organización de los datos es realizado por varios scripts que abren el archivo CSV y extraen los datos requeridos, aplicando las modificaciones necesarias para generar las instrucciones finales, las cuales son enviadas directamente al robot humanoide NAO para su ejecución.

En general, el sistema de control se puede separar en módulos que cumplen tareas específicas (descritas en la sección 3.1), y por tanto, para mejorar las capacidades del sistema se puede partir con el mismo modelo creando nuevos módulos que se comuniquen con el principal que ejecuta el movimiento en el robot NAO, considerando que este pueda requerir alguno que otro cambio.

Sistema controlado

Entiéndase como el robot humanoide NAO, quien recibe del sistema de control una serie de instrucciones con las acciones a ejecutar. El resultado esperado es la imitación de los movimientos del operador, considerando limitaciones físicas del robot y limitaciones de acción del sistema de teleoperación.

2.2. Herramientas de trabajo

A continuación el detalle del funcionamiento, limitaciones y requerimientos de las herramientas utilizadas para el desarrollo del proyecto.

2.2.1. Sistema de captura óptica de movimiento



Figura 2.2: MoCap en las instalaciones del PRIS-Lab, área con solo 8 cámaras, tomado en Noviembre 2017

El PRIS-Lab tiene en su posesión un conjunto de cámaras de rastreo óptico de la serie *Prime 41* (ver Figura 2.3) y el software de captura de movimiento *Motive*, ambos de OptiTrack, como sistema de captura de movimiento. Para las tomas se utilizó 8 cámaras ubicadas en distintas posiciones (ver Figuras 2.2 y 2.4, 2.5), el área visible en conjunto por las cámaras conforma el espacio de grabación donde el individuo debe realizar los movimientos deseados, proyectado a un espacio virtual 3D.



Figura 2.3: Cámara de la serie Prime 41.
Extraído de: <http://optitrack.com/products/prime-41/>

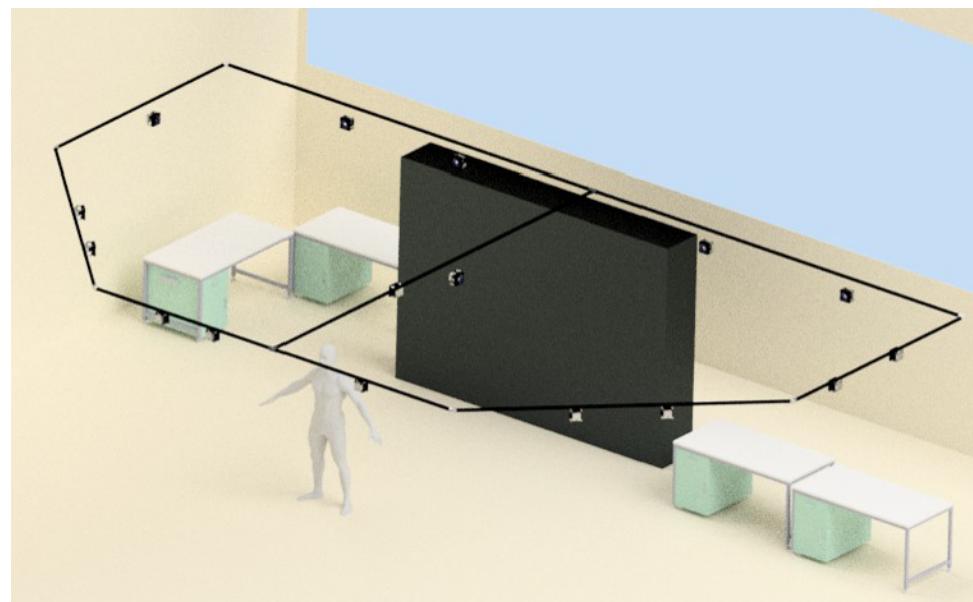


Figura 2.4: Configuración física del MoCap en el PRIS-Lab. [45]

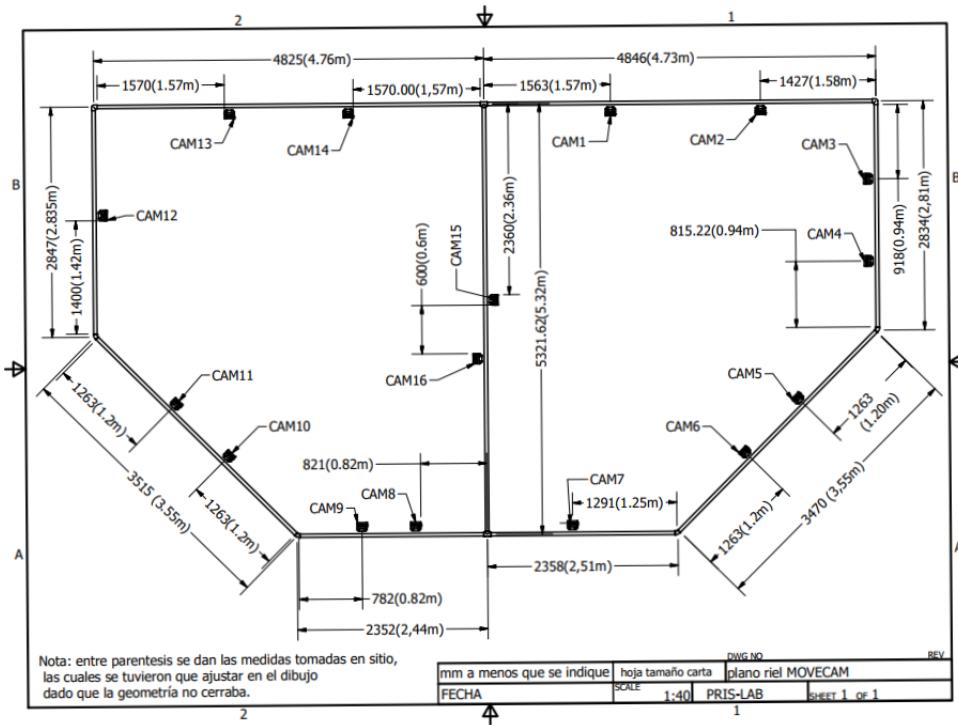


Figura 2.5: Plano del espacio físico disponible del MoCap instalado en el PRIS-Lab. [45]

Este sistema captura el movimiento del sujeto por medio del rastreo de unos marcadores especiales que se colocan sobre el sujeto en estudio en los puntos específicos de interés (brazos, piernas, etc). Estos marcadores (ver Figura 2.6b) son de forma esférica y se presentan en diferentes tamaños, con una cubierta retro-reflectiva que refleja la luz incidente, esta luz reflejada es la que las cámaras capturan. Los marcadores se colocan sobre el sujeto y son adheridas por velcro a un traje de tela (Ver Figura 2.6a). La intención del traje especial es asegurar que el sujeto quede cubierto de modo que solo los marcadores reflejen luz que incida en la persona, y facilitar la colocación de los marcadores.

Además, el traje es ajustado al cuerpo de la persona para proporcionar un pequeño margen de error en movimientos no deseados del marcador por el corrimiento de la ropa en medio de una coreografía, que provocaría que aunque la persona esté quieta el sistema de captura interprete que los marcadores están en vibración.



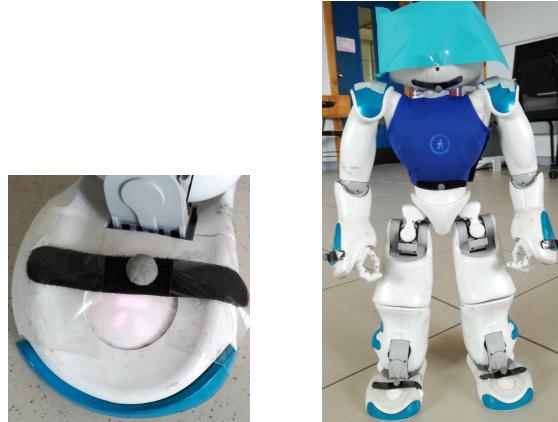
(a) Traje especial para las grabaciones en MoCap, con algunos marcadores colocados



(b) Marcadores del MoCap, grande y pequeño

Figura 2.6: Traje y marcadores para realizar grabaciones por MoCap

Cuando se desea capturar el movimiento de otros objetos en los que el velcro no se adhiere, los marcadores se deben colocar por algún método alternativo, tomando en cuenta que el material que se utilice no sea altamente reflectivo, ya que afectará la captura. En el caso del robot humanoide NAO se recurrió a utilizar cinta adhesiva para colocar los marcadores (Figura 2.7a), y se cubrió ciertos emisores LED de su cuerpo con papel opaco, de modo que la luz proveniente de estos no interfiriera con la toma de datos (Figura 2.7b).



(a) Marcador del MoCap colocado sobre pie del robot NAO, sostenido por cinta adhesiva

(b) Robot NAO con emisores LED de los ojos y cabeza cubiertos para que no interfieran en las tomas

Figura 2.7: Preparación del robot NAO antes de iniciar una grabación con el MoCap

Las cámaras son reconocidas por el software Motive, por medio del cual se realizan los ajustes necesarios para la obtención de los datos. La generalidad de los pasos requeridos para efectuar una captura de movimiento incluyen la aplicación de una máscara para ignorar reflejos permanentes que causan ruido en el área de grabación, la calibración de las cámaras, creación del espacio de captura por medio de la colocación del marco de referencia, introducción de los marcadores al espacio de captura (individuo con el traje y marcadores), y grabación del movimiento. Es importante que durante cada paso solo sean detectados los marcadores necesarios para su cumplimiento. Para mayor detalle en cómo efectuar este procedimiento es recomendable leer las indicaciones en las secciones *Quick Start Guide* y *Motive Workflow Pages* del Wiki de documentación de OptiTrack [42].

A continuación se enlista el equipo utilizado:

- 8 cámaras serie Prime 41.
- Vara con marcadores para la calibración del espacio de grabación (Figura 2.8a).
- Escuadra con marcadores para colocar el plano de referencia (Figura 2.8b).
- 6 ó más marcadores para el individuo (según el modelo utilizado, ver sección 2.3).
- Traje especial para colocar los marcadores.
- PC con software Motive y Licencias físicas para su uso.
- Cinta adhesiva.



Figura 2.8: Vara y escuadra para calibrar el MoCap

Extracción de los datos

Los marcadores rastreados son reconocidos por el software Motive, donde es posible observarlos como puntos en el espacio virtual generado. Los puntos se pueden interpretar de distintas maneras según el uso que se les quiera dar, en los párrafos siguientes se entra en detalle con esas variaciones. Basándose en el modelo más sencillo definido para la implementación de este proyecto, estos puntos corresponden a cada uno de los actuadores finales que se desean controlar en el robot humanoide NAO (ver Figura 2.15a). Mayor detalle sobre los modelos propuestos en la sección 2.3.

Luego de realizada la grabación, las diferentes tomas realizadas quedarán disponibles para edición y etiquetado de los marcadores dentro del proyecto creado en Motive, pero permanecerán en formatos exclusivos del programa. Es posible exportar la información del rastreo de los marcadores en varios formatos más comunes y fáciles de manipular, de ellos los más cómodos son el BVH y el CSV, cada uno con sus ventajas y complicaciones. Para mayor detalle en las especificaciones de la presentación de los datos en cada formato se recomienda revisar la documentación en línea http://v110.wiki.optitrack.com/index.php?title=Data_Export.

El formato BVH [30] [36] ofrece una organización jerárquica de los distintos marcadores involucrados, según un modelo de esqueleto previamente definido y adquirible según la versión de Motive. En este modelo se declaran ***huesos*** que conforman la estructura del cuerpo esquelético y un ***hueso raíz*** (generalmente cóxis) del cual se ramifican las demás partes del cuerpo y a partir del cual se realiza la ubicación espacial de los demás huesos. El hueso raíz es el único cuya ubicación espacial se da de manera *absoluta* con respecto al marco de referencia global. Es un formato muy organizado y que involucra una serie de transformaciones (relación entre cada hueso y la cadena de huesos que le conectan hasta la raíz), de este archivo es posible obtener hasta 6 DoF en ubicación espacial de cada miembro del

esqueleto. Estas cualidades lo hacen atractivo en la mayoría de aplicaciones de captura de movimiento. Sin embargo, es un formato que genera archivos extensos con un gran encabezado que expone la estructura jerárquica y las transformaciones que llevan hasta cada hueso, cada grabación distinta tiene transformaciones específicas para los datos capturados.

Además, Motive ofrece diferentes modelos de esqueletos preestablecidos para ser utilizados en las grabaciones, estos varían en área del cuerpo humano que cubren (según el área que se desea estudiar) y en cantidad de marcadores. Por ejemplo, el modelo de cuerpo completo de la Figura 2.9, el cual requiere de 49 marcadores para poder generar el modelo de esqueleto humano, un marcador ausente y se pierde el modelo; existen también modelos que generan un objeto tridimensional incluyendo sólamente Torso, brazos y cabeza, requiriendo 22 marcadores.

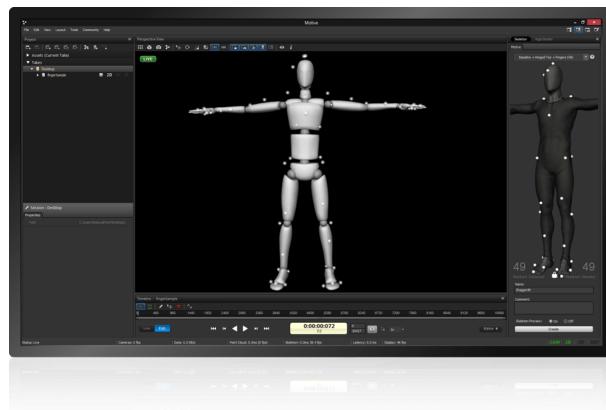


Figura 2.9: Modelo de cuerpo completo de 49 marcadores visto en Motive.

Extraído de: <https://optitrack.com/motion-capture-movement-sciences/>

Las desventajas de utilizar un modelo de estos es el requerimiento de cantidad de marcadores, que complica la captura debido a posibles pérdidas de la estructura del modelo y por tanto pérdida de información, y de la cantidad de objetos generados para formar toda la jerarquía del esqueleto completo, que implica tener que encontrar cuál objeto se ajusta mejor al actuador que se desea utilizar.

Para poder exportar los datos en la jerarquía del formato BVH se requiere tener un modelo de cuerpo esquelético establecido de antemano, el cual parte de la creación y encadenamiento de diferentes cuerpos rígidos.

Un cuerpo rígido [43] es creado en Motive a partir de tres o más marcadores que delimitan el espacio 3D del cuerpo que se quiere representar (por ejemplo uno de los huesos bases para formar un esqueleto completo). Este modelo permite la obtención de 6 DoF para cada objeto creado, lo cual es ventajoso ya que además de la ubicación del objeto y su delimitación en el espacio, permite conocer la orientación de su marco de referencia, esto es, rotación del objeto. Es posible exportar la información de un cuerpo rígido sin tener que generar un cuerpo esquelético, sin embargo, para que sea reconocible el cuerpo rígido a lo largo de la toma, se debe considerar que sus dimensiones no van a variar en el tiempo, es decir, la relación de espaciamiento entre los marcadores no varía en la toma, de lo contrario se pierde la figura y su información en el momento en que no se cumple esta proposición (Motive incluye un

margen de aceptación de este criterio, sin embargo es muy pequeño y por tanto es muy fácil que se pierda el rastreo del cuerpo rígido si el marcador no está bien colocado o el traje especial está muy flojo). Por ejemplo, un movimiento tan simple como abrir y cerrar la mano podría ocasionar que un marcador se mueva y en algunas tomas se pierda el cuerpo rígido.

Esto supone un problema en algunas tomas ya que si alguno de los marcadores que conforman el cuerpo rígido se mueve de modo que no se respete la constancia en alguna de sus dimensiones, Motive no lo reconocerá como cuerpo rígido y se perderá la información de los cuadros en que esto sucedió. Además, es posible que el sistema no sea capaz de diferenciar entre dos cuerpos rígido independientes entre sí (que no forman parte de una cadena de objetos) con la misma forma, para ello se requiere determinar un patrón de colocación de los marcadores diferente entre los cuerpos rígidos que permita delimitar correctamente el objeto y diferenciarlos entre sí, esto permite además que si se pierde el cuerpo rígido en unas tomas, cuando se vuelva a cumplir con su estructura será reconocido automáticamente de nuevo.

Además del BVH es posible exportar la información en formato CSV, el cual presenta un encabezado más simple que el BVH y los datos se exponen de manera directa con su ubicación espacial absoluta respecto al margen de referencia del espacio de grabado (o bien respecto a algún otro punto deseado), no se debe utilizar transformaciones específicas para obtener la ubicación individual cartesiana de cada objeto. Esto es ventajoso ya que permite usar modelos de solo marcadores o solo cuerpos rígidos (que no lo permite un BCH) sin la necesidad de crear un modelo esquelético, sin embargo, se pierde la organización que ofrece el BVH para identificar cada objeto y la posibilidad de generar un modelo 3D del cuerpo completo y la relación entre sus partes. Este camino presenta simpleza en el análisis y procesamiento de los datos, ventaja que se aprovecha para ejecutar pruebas en etapas tempranas del desarrollo del proyecto para estudiar el comportamiento del sistema de teleoperación y así mismo definir los futuros requerimientos para un sistema que incluya 6 DoF, así como ver alternativas al BVH y los modelos esqueléticos, utilizando solo cuerpos rígidos no encadenados, o marcadores individuales.

En el caso de exportar la información de cada marcador de manera individual en CSV (disminuye la cantidad de marcadores a solo 1 por sección a rastrear), es posible que su organización en archivos de tomas distintas no sea el mismo (orden de aparición de las etiquetas de los marcadores y sus respectivos datos), lo cual requiere de agregar una capa de acomodo de los datos, en caso de ser necesario un orden específico. Esto porque cada toma, aunque en un mismo proyecto, es independiente y puede existir ruido en el ambiente que en una toma anterior no existía (por ejemplo un destello que provoque la inclusión momentánea de un nuevo marcador y por tanto de nuevas etiquetas automáticas). Las etiquetas que pone el sistema no corresponden a nombres fácilmente reconocibles para su ubicación en la figura que se está grabando, ni con nombres especiales que puedan ser utilizados por código en fases siguientes del proceso de teleoperación. Por tanto, es importante identificar y etiquetar los marcadores de la figura humana, para poder reconocer el actuador al que corresponde la información del marcador. En el caso de los cuerpos rígidos es posible etiquetarlos previo al inicio de las grabaciones de modo que al iniciar con el proceso de grabación ya se incluya sus nombres en cada grabación realizada. Para un modelo de esqueleto humano los huesos ya tienen su etiqueta automática que identifica qué parte del cuerpo representa, según el mismo modelo.

Al exportar la información de los marcadores individuales se obtiene solo 3 DoF (XYZ), dado que son considerados como puntos en el espacio, por lo que es importante considerar el uso de cuerpos

rígidos o un modelo esquelético para refinar la ubicación de los actuadores del NAO. Cabe destacar que de igual manera se requeriría la definición y restricciones del cuerpo rígido y los modelos del esqueleto para que sean detectados como tales en las tomas, lo que escalaría a mayor cantidad de marcadores.

Para incluir los datos de las rotaciones se utiliza un modelo de esqueleto humano con 42 marcadores (ver Figura 2.10). Para poder seguir utilizando el planteamiento de 6 cadenas de actuadores correspondientes en el NAO se extrae la información de ubicación espacial para 6 DoF de los COM de las partes del cuerpo del modelo 3D que se asemejen al actuador final de la cadena del robot, es decir, **RArm** y **LArm** se controlan con las manos del operador, **RLeg** y **LLeg** con los pies, **Head** con la cabeza y **Torso** con el Tórso. Estos COM, al corresponder a cuerpos rígidos dentro del modelo esquelético, incluyen la información de rotación que se ocupa para refinar el movimiento del robot, ya que con solo 6 marcadores totales se puede estudiar solamente la dirección del movimiento, pero con la extracción de información de 6 DoF ya es posible hacer un estudio de ubicación espacial con inclinación.

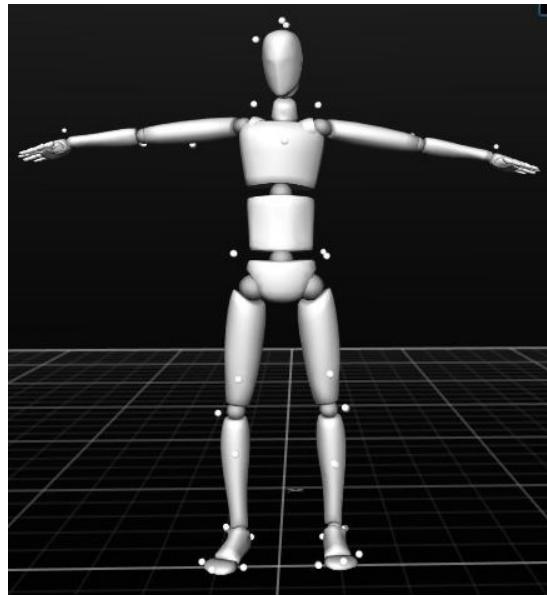


Figura 2.10: Modelo de cuerpo completo de 42 marcadores en Motive.

Es importante resaltar que en las propuestas anteriores se requiere realizar grabaciones y generar un archivo en el cual se almacena la información, ninguno de estos métodos permite el control *en tiempo real*.

Motive permite la transmisión de datos sin tener que recurrir a estos archivos, por medio de la arquitectura NatNet [40], donde desde Motive se crea un servidor que envía los datos por la red, y es posible crear un cliente en un computador aparte que reciba los paquetes con la información, esto añadiría una etapa extra al ajuste de los datos que conforma el desempaqueado de la información transmitida y su acople para entenderle como coordenadas espaciales relacionadas a un identificador.

El tema de la transmisión en tiempo real no es tratado en detalle en el presente documento, ya que el enfoque del proyecto es el uso del archivo CSV, por la necesidad de independizar las pruebas

y el desarrollo del acceso constante al área del MoCap y a las licencias de Motive, así como para la aplicación del estudio de movimiento humano en grabaciones y en el *entrenamiento* del NAO para que juegue fútbol de manera autónoma, es decir, sin control en tiempo real. Para mayor información se recomienda revisar la documentación oficial [38] [41] y revisar la información de foros y proyectos relacionados al uso de NatNet y su funcionamiento en Linux [39].

2.2.2. Robot Humanoide NAO

Softbank tiene una serie de robots con anatomía humanoide, cuya finalidad es que se mezclen con mayor facilidad dentro de la vida cotidiana de las personas y el ambiente urbano diseñado para las personas, con respecto a robots con otras apariencias. Dentro de estos robots se incluye los modelos NAO, un robot con apariencia infantil con estatura promedio de 57 cm [50]. El proyecto hace uso de un robot NAO cuya anatomía corresponde al modelo académico H25 (ver Figura 2.11). Posee gran cantidad de sensores y actuadores, ofreciendo 26 DoF de movimiento, lo que complica el control estable de su movimiento. Para fines del proyecto se aplican limitaciones de operación que varían según se avanza en el desarrollo de la aplicación. Para ver más detalles se recomienda revisar la documentación oficial en línea que ofrece ALDEBARAN, como la guía de usuario [46].

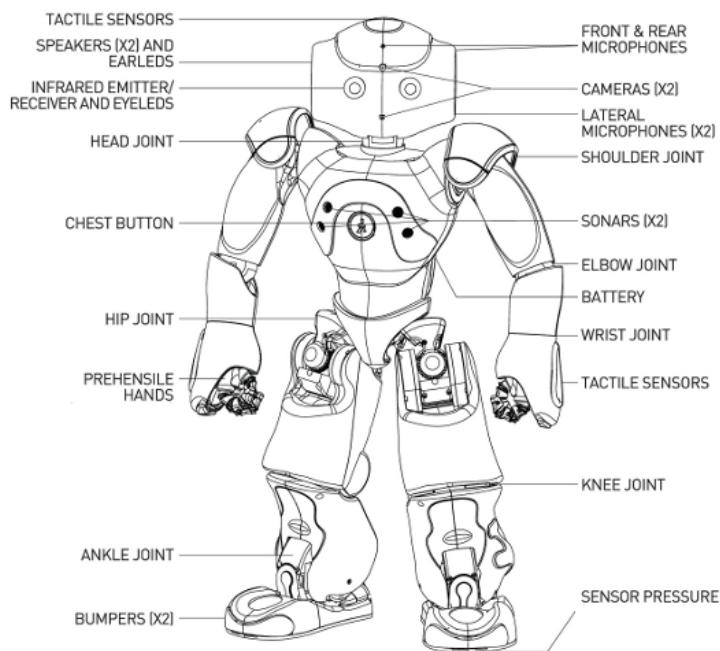


Figura 2.11: Configuración Física del Robot Humanoide NAO modelo H25.
Extraído de: http://doc.aldebaran.com/1-14/family/nao_h25/index_h25.html

Es importante mantener al robot siempre sobre una superficie plana y en un ambiente seguro, esto para prevenir daños que puedan surgir de caídas inesperadas. Dada su fácil inestabilidad al ejercer diferentes movimientos, es común que se den muchas caídas.

El robot NAO puede conectarse por WiFi, para lo que se necesitará una red configurada en la cual se conecten el NAO y el computador encargado de enviar las instrucciones, o bien directamente por cable de red entre computador y robot.

Para ejecutar el código se necesita tener acceso al SDK correspondiente al lenguaje de programación a usar (C++ o Python), ya que esto incluye las herramientas de desarrollo de software necesarias y permite la creación de objetos con los cuales utilizar las instrucciones de control intrínsecas del robot desde los distintos API's. Los principales API's utilizados son ALMotion API [2] y ALRobotPosture API [3]. Es recomendable estudiar estas herramientas desde la documentación oficial de ALDEBARAN para saber cómo implementar sus funciones, requisitos y limitaciones de la operación, y su relación con las diferentes acciones que se quiere que el robot NAO ejecute.

ALRobotPosture API

Este módulo permite controlar el robot NAO de modo que su cuerpo tome una de las posiciones predefinidas. De este módulo se utiliza la instrucción *goToPosture()*, dado que genera una trayectoria de manera automática para ir de la posición en que se encuentra al momento del llamado hacia la posición deseada, y permite el ajuste de la velocidad con que se ejecutará el movimiento.

Esta herramienta es importante dado que permite de manera sencilla poner al robot NAO en posiciones estáticas seguras, ya sea como preparación para realizar alguna próxima acción, para algún punto intermedio dentro de la rutina en que sea necesaria una pausa segura, o bien al finalizar la acción.

La lista de posiciones disponibles se encuentra en la documentación en línea [12]. De estas posturas se recomienda el uso de **Stand** (Figura 2.12a) como pose previa al control del movimiento del NAO, dada su estabilidad estando de pie y que en general la persona en el área del MoCap estará de pie con una posición equivalente en momentos de reposo o para iniciar la animación, de este modo se facilita las primeras transiciones de posición del NAO. Se recomienda además utilizar una de estas poses al finalizar la rutina de movimientos, un ejemplo es la pose **Crouch**, que además de dar mayor seguridad a caídas que estando de pie, ofrece una forma cómoda para trasladar el NAO e indicar que está inactivo.

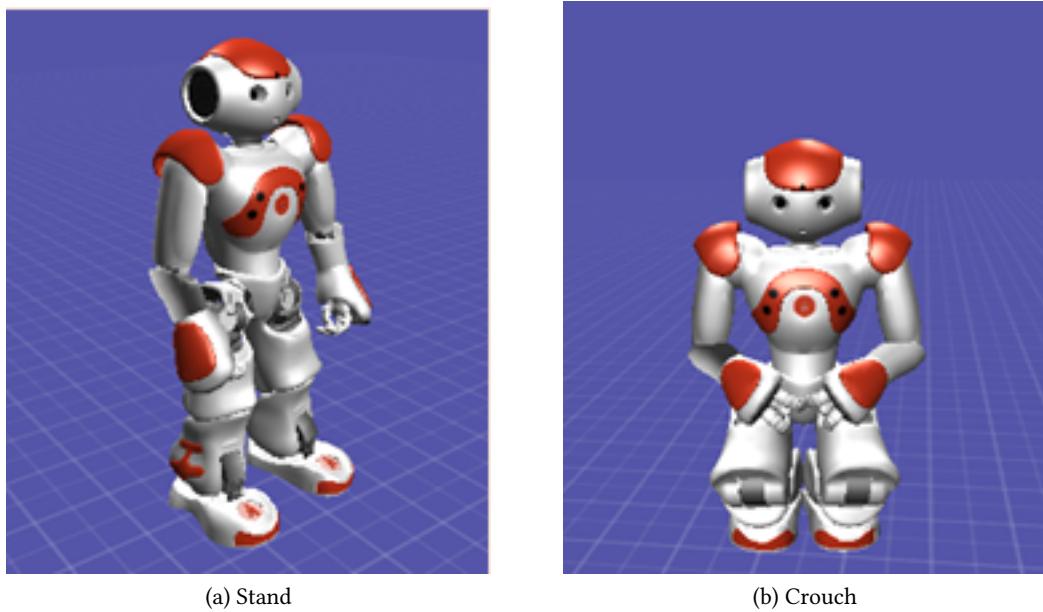


Figura 2.12: Posiciones predefinidas recomendadas.

Extraído de http://doc.aldebaran.com/2-1/family/robots/postures_robot.html

ALMotion

Este modulo provee de métodos que facilitan el control del movimiento del robot [2] [4]. Se divide en 4 grupos principales según el control deseado:

- Rrigidez de las articulaciones
- Posición de las articulaciones
- Caminado
- Actuadores en el espacio Cartesiano

De estos, los más importantes para el proyecto son el control de la rigidez de las articulaciones y el control de los actuadores en el espacio cartesiano.

En cuanto a la rigidez de las articulaciones, su importancia radica en que si no se activa la rigidez, no es posible mantener una posición fija ni cambiar de posición los actuadores, de modo que al inicio de toda rutina que requiera que el robot se mueva es necesario activar la rigidez del mismo, ya sea de algunos actuadores en particular, o bien del cuerpo completo (*Body*). Hay que considerar que tener la rigidez activa calentará los motores con el tiempo, por lo que al finalizar toda rutina es necesario desactivar la rigidez para liberar los actuadores y que *descansen*. Tomar en cuenta que al momento de desactivar la rigidez los motores se moverán libremente, para evitar caídas se debe primero llevar al

NAO a una posición estable y segura donde pueda permanecer sin los motores activados, como sentado o acostado, y luego desactivar la rigidez de los motores.

En cuanto al control de los actuadores en el espacio cartesiano, es lo que se pretende con este proyecto a partir de los datos del MoCap. Esta sección incluye diferentes maneras de enviar la posición deseada y vectores de movimiento al NAO, así como habilitadores del movimiento de cuerpo completo.

Para mover al NAO con sus actuadores hay que tomar en cuenta varias consideraciones. Es posible indicar una pose general al NAO por medio de la ubicación espacial de cada uno de sus 13 actuadores principales que conforman 26 DoF del NAO(Figura 2.13a), pero se requeriría de mucha información y un sistema de control muy detallado que acople cada cadena de actuadores de modo que se vean movimientos no forzados y estables. Además, se necesitaría obtener información para cada uno de los actuadores, lo que implica mayor cantidad de marcadores en el modelo del humano bajo el MoCap. Nótese en este punto la aplicación de un modelo 3D de esqueleto humano para ejercer un control más complicado pero detallado sobre el NAO, con cada actuador o articulación independiente.

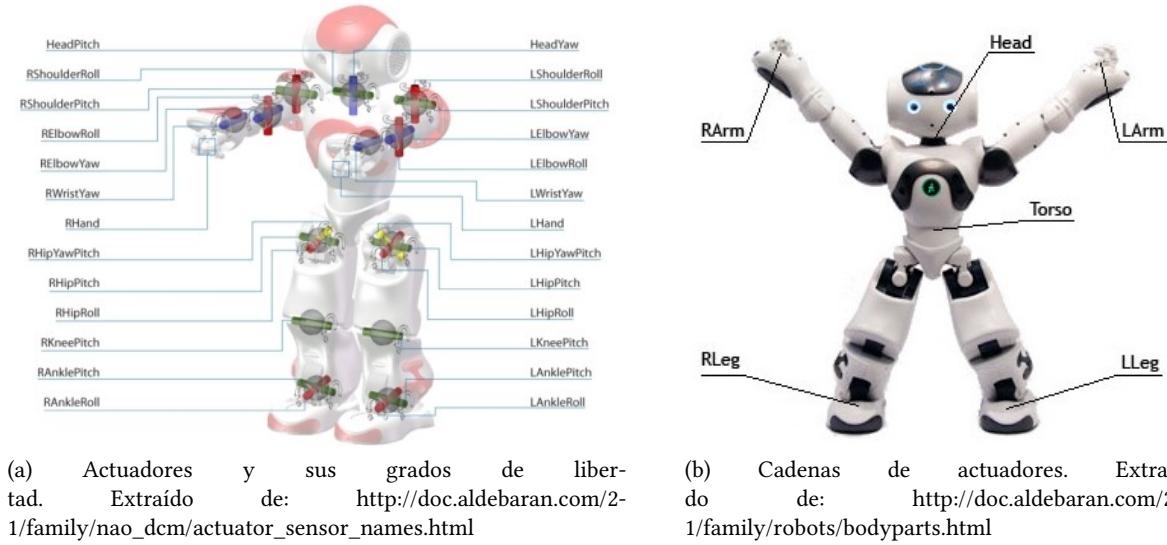


Figura 2.13: Actuadores y Cadenas del robot humanoide NAO.

Para simplificar el caso de controlar cada uno de los actuadores, el NAO incluye un proceso de ubicación automática y encadenamiento, donde se agrupan en cadenas⁴ de actuadores finales⁵ [6] (ver Figura 2.13b, lo que reduce la cantidad de motores sobre los que se ejerce control, y además garantiza mayor *naturalidad* en los movimientos observados. Estas cadenas de actuadores se asemejan a la jerarquía del modelo del esqueleto humano. Para lograr esto se usa un modelo de 6 cadenas, donde la información importante para el movimiento es la ubicación espacial del actuador final de la cadena y la rotación espacial del marco de referencia local de cada uno, estos datos con respecto a un marco de

⁴Del inglés (*effector*) *chains* para referirse al conjunto de articulaciones (actuadores, motores) que conforman una extremidad completa o conjunto móvil del NAO.

⁵Del inglés *End Effectors* para referirse al actuador final de la cadena de actuadores

referencia definido. En la documentación oficial del robot se detalla la definición de los actuadores y las cadenas, su operación y limitaciones físicas según el modelo del cuerpo del robot usado [9].

Al ver los actuadores a controlar según el modelo de actuador final, cada objeto (actuador final) puede ser ubicado espacialmente con máxime 6 DoF, de modo que la información obtenida en formato CSV (y BVH) es compatible para su control. También es posible trabajar con menores DoF por medio de las combinaciones de los diferentes puntos de control, X, Y, Z, wX, wY y wZ (*w* indica rotación sobre el eje).

Para ubicar los actuadores en el espacio se debe definir un marco de referencia respecto al cuál se dan las coordenadas de ubicación. El NAO presenta 3 diferentes marcos a utilizar, cada uno puede ser aprovechado de distintas maneras según el enfoque del movimiento deseado: **ROBOT**, **TORSO** y **WORLD**. De ellos, se usan *Robot* y *Torso* para el desarrollo del proyecto (Figura 2.14).

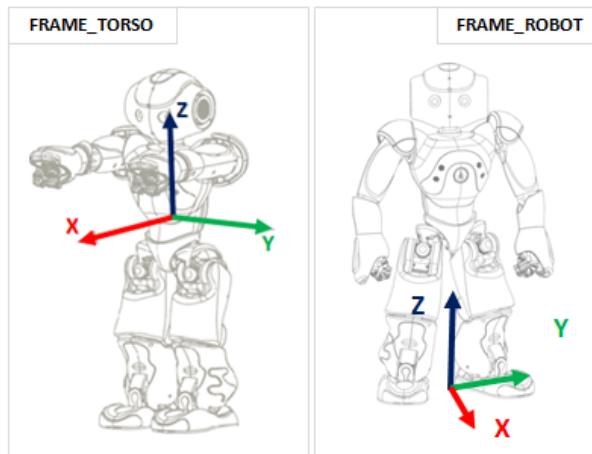


Figura 2.14: Marcos de referencia TORSO y ROBOT.

Extraído de: <http://fileadmin.cs.lth.se/robot/nao/doc/naoqi/motion/control-cartesian.html>

El marco *ROBOT* se origina bajo la aproximación del punto medio entre los pies del NAO y sobre el plano de apoyo de los mismos. Esta referencia es útil para coordenadas absolutas respecto a un marco de referencia externo que coincide con la ubicación de este, entre las piernas y por debajo del Torso. Este coincide a colocar la escuadra de ajuste del suelo en MoCap debajo de la persona que se va a mover. Dentro de las limitaciones se puede mencionar que no permite el control cartesiano directo de los pies, ya que de la ubicación de estos es que surge el marco de referencia, es decir, si los pies se mueven así mismo lo hará el marco de referencia. Cabe destacar que simplifica el control del torso y de los brazos.

El marco de referencia *TORSO* se ubica en el centro de masa de la sección del Torso del robot NAO (Figura 2.19d). Esta referencia es útil ya que permite una relación anatómica entre las cadenas de actuadores, todas referenciadas al Torso, y permite el movimiento de los pies. Sin embargo, si el marco de referencia del MoCap no coincide con este, se debe ajustar los datos al traslado de referencia. Además, hay que considerar que ya que se permite el movimiento libre de las piernas, es más fácil incurrir en una posición inestable para el robot, para ello se requiere de un control más detallado del balance. El control directo de los pies es considerado como control complejo del NAO.

Es importante destacar que las funciones para el control del movimiento de los actuadores del NAO pueden operar de dos maneras distintas. La primera es de manera bloqueante (*métodos de animación*), es decir, al realizar el llamado y ejecución de la función se bloquea mediante una bandera ciertos accesos al control de los actuadores, y estos son liberados hasta que termine la ejecución del llamado. Si esta ejecución tarda mucho, la demora en comenzar con la siguiente instrucción puede provocar una percepción de torpeza en los movimientos del NAO, por lo que hay que cuidar su uso. Estos llamados no bloquean operaciones *transparentes* al usuario como los comportamientos de manejo de caídas o bien el balance automático del NAO. Pero funciones de acceso por usuario sí son bloqueadas, por ejemplo si se quiere reproducir audio mientras se realiza la acción bloqueante⁶. Un ejemplo es la función ***positionInterpolations()***, que permite controlar las 6 cadenas al mismo tiempo por medio de un set de vectores de coordenadas para cada cadena en un llamado bloqueante. Sirve para enviar una grabación de movimiento completa para que el NAO la ejecute, sin embargo, no se puede ejercer ningún control directo sobre los actuadores en medio de la ejecución, como cambiar las trayectorias.

La segunda manera es por llamado no bloqueante (*métodos reactivos*), que permite la ejecución de otras funciones de control directo. Sin embargo, su operación es más limitada y no permite el control del cuerpo completo, sino por cadenas individuales de actuadores. Este llamado tampoco interfiere con procesos transparentes al usuario. Un ejemplo es la función ***setPosition()***, que permite controlar el posicionamiento de una cadena de actuadores de manera no bloqueante, esto permite cambiar la trayectoria deseada en medio de la ejecución del movimiento, sin embargo, para mover el cuerpo completo se requeriría de un llamado independiente para cada cadena, que a su vez incluye un pequeño desfase en el inicio del movimiento de los actuadores según el orden en que ejecuten los llamados. El solucionador del posicionamiento y de movimiento del NAO requiere aproximadamente 20 ms para preparar una acción de cinemática, independientemente de cuánto tiempo tarde el robot en moverse.

Es posible tomar una de estas acciones de control de movimiento, que en general suelen ser más largas y lentas que otras acciones del NAO, y enviar la instrucción como una tarea de movimiento⁷ y ejecutar otra acción en paralelo, por ejemplo desplazar al NAO mientras reproduce audio, por medio del atributo 'post' incluido en cada proxy creado. Esto indica que la ejecución de la instrucción se debe realizar en segundo plano, permitiendo que el código continúe con la siguiente instrucción. También es posible esperar a que se complete alguna de estas tareas enviadas a segundo plano y detener el código, por medio de etiquetas generadas al utilizar 'post' [11].

El módulo además ofrece una serie de instrucciones para controlar diferentes aspectos del robot que se ejecutan de manera paralela (en la mayoría de los casos) a la rutina de control de movimiento directo, definidos antes de la ejecución del movimiento, esto para facilitar el control de cuerpo completo del robot [14]. Estas son operaciones transparentes al usuario. De ellas resulta bastante útil el **balance automático de cuerpo completo**.

⁶Es posible reproducir audio y mover al NAO por medio del control de las tareas de movimiento, para ello se usa el atributo *post* que permite enviar la tarea bloqueante a segundo plano y en *paralelo* seguir con los llamados del usuario como reproducir audio.

⁷Motion Task

Balance de cuerpo completo

Se lleva a cabo por medio de una serie de instrucciones del API de control de cuerpo completo [15], las cuales permiten activar/desactivar el manejo automático del balance, seleccionar ciertas restricciones para el balance y mover el centro de masa de manera automática. Esto permite que el resultado del movimiento se vea más natural y sea un poco más seguro ante posibles caídas.

La solución de estabilidad es realizada mientras la tarea de balance esté habilitada, resolviendo la simplificación cuadrática de un problema de Cinemática Inversa Generalizada [20] cada 20 ms (mayor detalle en la documentación [14]) considerando las restricciones configuradas por medio del API, así como restricciones físicas conocidas por el NAO, llevando al robot a la posición posible más próxima. Sin embargo, dependiendo de las transiciones y posturas que realice el NAO se puede llevar al COM fuera del polígono de soporte definido para el balance y esto hará que se active el **Administrador de Caídas** al llegar a una posición peligrosa, con lo que se interrumpe el proceso deseado.

La interrupción del Administrador de Caídas [7] es afrontado inhabilitando esta función. Este administrador está dedicado a detectar si el COM llega a salirse del polígono de soporte, generando una *Tarea* para responder ante el *Evento* de la caída, que llega a tomar alta prioridad dentro de la lista de tareas en ejecución, bloqueando todo proceso de control de movimiento actual, es decir, las tareas del control remoto sobre los actuadores que se estaba ejecutando son terminadas y el programa externo deja de correr sobre el robot. Luego de la interrupción, se activan las rutinas de protección ante caídas, que en general se puede describir con los siguientes pasos: cubre cabeza y torso con sus brazos, extiende los pies, deshabilita la rigidez en los actuadores para agregar amortiguación en la caída, espera estabilidad luego de haber caído, *reconoce* posición actual y busca llegar a la pose predefinida más cercana a su estado caído por medio una rutina guardada en su memoria (busca llegar a posturas muy seguras como **Crouch** o **SitRelax**). Este proceso, aunque útil y protectivo, resulta molesto al ejercer el control que se desea, en que el COM muchas veces estará en zonas de riesgo en especial durante pruebas de movimiento.

Para inhabilitar este reflejo se necesita primero habilitar su desactivación (ver documentación [7] y detalles en la sección 3.7.4) en el NAO accediendo a la configuración de desarrollo, y luego habilitar/inhabilitar el administrador de caídas cuando se requiera [8]. De este modo cuando el robot se acerque a una posición inestable, pero necesaria para las pruebas, no se activará el reflejo de caída para interrumpir con el proceso en desarrollo. Es importante tomar en cuenta que ya que se ha inhabilitado este administrador, se debe tener mayor vigilancia ante caídas del robot, ya que no tratará siquiera de protegerse al caer. Es recomendable designar a una persona exclusivamente a la protección del NAO durante pruebas que involucren su movimiento.

En cuanto al desbalance durante las tareas de balance automático, mientras esta tarea esté activa el NAO tratará de cumplir con las restricciones asignadas por las instrucciones del API. Estas restricciones se conforman de dos indicaciones: *Actuadores Restringidos* y el *Espacio de Restricción*. La indicación de los actuadores está definida por **Legs**, **LLeg** y **RLeg**, indicando: ambas piernas, pierna izquierda y pierna derecha respectivamente, donde el primero da la misma restricción a ambas piernas, y el segundo y tercero permite configurar la restricción para cada pierna de manera independiente. El espacio de restricción corresponde a los DoF admisibles para el movimiento del actuador restringido, pudiendo ser estos **Fixed**, **Plane**, **Free**, donde el primero restringe el movimiento para los 6 DoF del actuador, el

segundo restringe al actuador en el plano sobre el que se apoya, permitiendo desplazamiento en X e Y y rotación al rededor del eje Z (Figura 2.19c), y el tercero permite que el actuador se mueva libremente en todo el espacio cartesiano. La restricción de balance lo que indica al robot es el punto de apoyo para el movimiento y cómo se debe comportar respecto a ese punto, según la información descrita.

Las diferentes combinaciones de restricciones son aplicables a distintas situaciones de balance, de modo que la recomendación es adaptar la combinación necesaria según la situación, ya que si se mantiene permanentemente una única combinación, es posible que para alguna postura no sea suficiente, y a pesar de lograr el balance del robot, generará *meneos* y movimientos forzados en los actuadores de las piernas que pueden llegar a ser perjudiciales para la integridad de los motores, así como desbalancear al NAO y hacer que llegue a posturas incómodas para el administrador del balance y que este no pueda encontrar una solución viable o con una respuesta lo suficientemente rápida como para que el NAO no se caiga. Sin embargo, el uso de la restricción Fija (*Fixed*) en ambas piernas es un buen inicio para ciertas pruebas, en especial las que involucran solo el movimiento de los brazos, ya que el administrador bloqueará el movimiento cartesiano de los pies, que no se están usando y así no se debe preocupar mucho por su control (no así con las demás articulaciones en las piernas que sí se pueden mover para desplazar el resto del cuerpo, siempre que mantenga los pies en la posición fija).

Para mejorar el efecto de las restricciones de balance se debe escoger la combinación más adecuada según el movimiento que está ejecutando el NAO. Para ello se debe considerar la posición actual y la posición a la que se desea llegar. El administrador del balance ofrece una gran ventaja respecto a esas consideraciones ya que limita el movimiento del robot con respecto a sus capacidades físicas, de modo que si la información que recibe es, por ejemplo, estirar el brazo de modo que el COM se salga del apoyo sobre el cual se restringió el movimiento, no permitirá que suceda, la restricción de balance cobra mayor importancia en el movimiento. De modo que para el control del NAO, con el nivel de simpleza que se trata de lograr en este proyecto, no será necesario preocuparse por esas situaciones con datos fuera del rango de acción (al menos en cuanto a ejercer alguna corrección con base en un análisis de los datos con respecto al campo de acción del actuador), sin embargo, el NAO de igual manera recibirá la información y tratará de llegar al punto solicitado más cercano, de modo que si se le pasa información errónea el movimiento no corresponderá al esperado, se podría llevar al robot a posiciones complicadas en que el desbalance no sea corregido y por tanto el NAO se caiga o mueva sus extremidades a posiciones dañinas (por ejemplo, al confundir la información entre los brazos izquierdo y derecho, o pasar los datos de un brazo al torso). El filtro que asegure que se pasen los datos correctos será la persona encargada de la grabación en MoCap, y los respectivos procesos del sistema de teleoperación que acomoden los datos en los formatos y requerimientos seleccionados, partiendo que vienen bien desde el MoCap (ver capítulo 3). Sin embargo, el sistema de teleoperación puede ser mejorado con la inclusión de un módulo que haga ese análisis y corrección de las posiciones deseadas para los actuadores y los ajuste a conveniencia según la aplicación e intención del movimiento, mejorando el resultado de la ejecución final.

El API ALMotion incluye la instrucción ***wbGoToBalance()*** que permite enviar a discreción al COM sobre un punto de apoyo (Legs, LLeg, RLeg) en un tiempo determinado. Esto puede ayudar a ejercer coerción sobre el balance automático del NAO establecido desde antes del movimiento, para que sabiendo dónde está el COM, este cambie de apoyo según se considere más conveniente para una posición en específico. Sin embargo, hay que tener presente que es un llamado bloqueante (hay que esperar a que el NAO lleve al COM al apoyo para que se ejecute la siguiente transición de movimiento).

Además, existe la instrucción `getCOM()`, que devuelve la ubicación espacial XYZ del COM del NAO con respecto a un marco de referencia escogido (revisar Figura 2.14), con lo que es posible verificar si el COM se encuentra en peligro y modificar las coordenadas a ejecutar para ajustar su ubicación, por ejemplo, con el movimiento del Tórso.

Con la combinación de estas condiciones que permite el API ALMotion es posible generar un mejor balance para el robot NAO donde se conozca el COM sin realizar cálculos externos a los que ya realiza el robot con estas funciones, y con la capacidad de ubicarlo según condiciones de apoyo. Cabe resaltar que estas son funcionalidades básicas, si se quisiera emplear un control más robusto y completo donde se tenga mayor control sobre cada elemento de la operación será necesario siempre agregar más módulos especializados en esos detalles. Bien se podría balancear el NAO sin el administrador de balanceo automático, sino con algún módulo externo que calcule y proyecte la ubicación del COM del NAO y mueva los distintos motores del robot para posicionar el COM a como se quiera.

2.3. Ajuste Humano - NAO y modelos para la captura de movimiento

La anatomía humanoide del NAO permite establecer una relación directa con el modelo del cuerpo humano que hace que se pueda interpretar el movimiento de la persona como el movimiento final esperado por el NAO de manera sencilla. Sin embargo, esta información requiere de un ajuste dadas las diferencias en tamaños de una persona y el robot, así como efectos en el posicionamiento de las extremidades debido a la flexibilidad de un humano comparado con la rigidez de las partes que conforman el cuerpo del robot.

Para ejercer el control en el robot NAO y obtener los datos desde el MoCap se pueden utilizar tres modelos distintos, según se ha diseñado para este proyecto. El primero es uno sencillo de 6 puntos de control (círculos rojos de la Figura 2.15a) correspondientes a los actuadores finales de las cadenas de acción del robot humanoide NAO, y a los marcadores a colocar en la persona para la grabación en MoCap. Nótese que por la simpleza del modelo solo se consideran los actuadores finales de las extremidades, y el torso, de modo que puntos intermedios no son manipulados directamente por el sistema de control. La ventaja del modelo es la poca cantidad de puntos por los cuales preocuparse al ejercer la teleoperación, sin embargo, al ser solo puntos en el espacio no se incluye la información de las rotaciones en la ubicación de los actuadores, y por tanto el solucionador de posición de ALMotion del NAO podrá encontrar gran cantidad de soluciones posibles que lleven al actuador al punto espacial indicado, es decir, el resto de articulaciones en la cadena de acción controlada pueden moverse de muchas maneras que satisfagan la ubicación del punto y la solución ejecutada podría no corresponder con la figura de la extremidad de la persona a pesar de que se haya cumplido con el posicionamiento espacial del actuador de interés (ver Figura 2.16).

Por lo que si se desea mantener mínima la cantidad de marcadores requeridos por el modelo es posible utilizar la configuración de 6 puntos descrita, con solo un marcador por punto de acción, teniendo en cuenta entonces que el estudio del movimiento del NAO en reacción a la grabación del MoCap será únicamente para posicionamiento del actuador final correspondiente de cada cadena (**RArm**, **RLeg**, **LLeg**, **LArm**, **Torso** y **Head**) solo para coordenadas X, Y y Z.

El problema con las posibles soluciones generadas por el posicionamiento solo en (X,Y,Z) se puede

observar en las imágenes de la Figura 2.16. Se trata de ubicar la mano derecha de un sujeto en el mismo punto espacial con dos configuraciones distintas para el codo y el hombro derechos. Si se considera solo la posición en (X,Y,Z) ambas configuraciones (Figuras 2.16a y 2.16b) aunque diferentes satisfacen dicha ubicación, a como se observa en la Figura 2.16c, comparten la ubicación del centro del actuador final (punto rojo) a pesar de que el resto de articulaciones de la cadena de actuadores del brazo derecho están en posiciones diferentes. El NAO cumple con la tarea propuesta en ubicación 3D aunque la pose no concuerda a exactitud.

Al considerar este punto de control como un objeto (pasar de un modelo de solo 6 marcadores a uno de cuerpos rígidos con más marcadores por punto de acción), se adquiere información de su inclinación en el espacio, incluyendo (wX,wY,wZ) como datos para su ubicación espacial (en la Figura 2.16d se representa los ejes de rotación intrínsecos al COM del objeto 3D creado como cuerpo rígido). Con esto se logra que aunque concuerden en la ubicación (X,Y,Z), por el acomodo que deben hacer el codo y el hombro en cada una de las soluciones posibles, los datos de rotación (wX,wY,wZ) diferirán entre las posiciones, disminuyendo así las soluciones posibles para ubicar al actuador final en el espacio al eliminar aquellas en que no se satisface la rotación espacial. Al incluir rotación no se elimina las múltiples soluciones, pero se logra que las posiciones viables tengan un menor rango de variación entre sí para cada articulación involucrada en la cadena de acción.

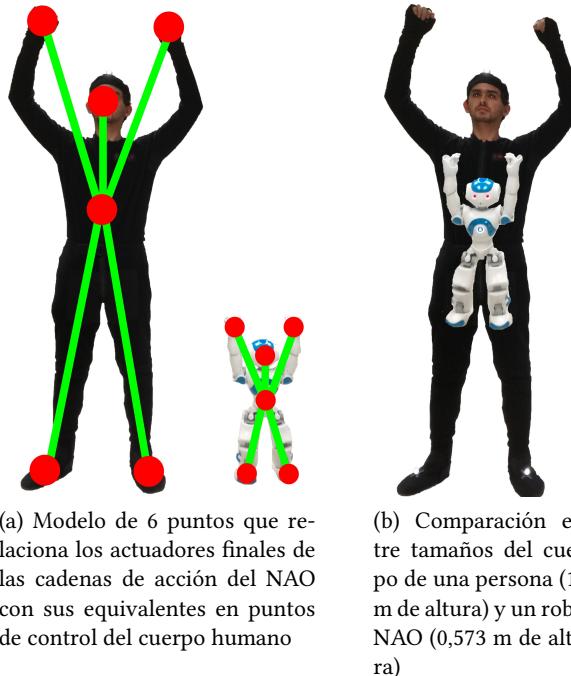


Figura 2.15: Comparación entre los cuerpos de una persona y un robot humanoide NAO H25

Por tanto, para reducir las posibles soluciones a la ubicación espacial del actuador es necesario

incluir la información de la rotación del elemento (Figura 2.16d, esto requiere la creación de cuerpos rígidos en Motive. Siguiendo con el objetivo de reducir lo más posible la cantidad de marcadores involucrados y mantener simple el modelo de relación humano-NAO, se puede utilizar pocos marcadores para crear un cuerpo rígido por actuador a controlar, en Motive se requiere mínimo 3 marcadores por objeto a crear. El modelo seleccionado, y segundo modelo propuesto, se deriva del cuerpo esquelético de Motive con 42 marcadores de la Figura 2.10. se eliminó los marcadores que no se ligaban directamente con los cuerpos rígidos a utilizar (muslos, antebrazos, entre otros) y se reacomodó los de las manos para que los movimientos de muñeca u de abrir y cerrar la mano no deformaran el cuerpo rígido, generando un modelo con 3 marcadores para cada mano y la cabeza, 5 para los pies y 8 para el torso, usando en total 27 marcadores (ver Figura 2.17, en las fotografías los marcadores se ven brillantes debido a la luz que reflejan por el *Flash* de la cámara.).

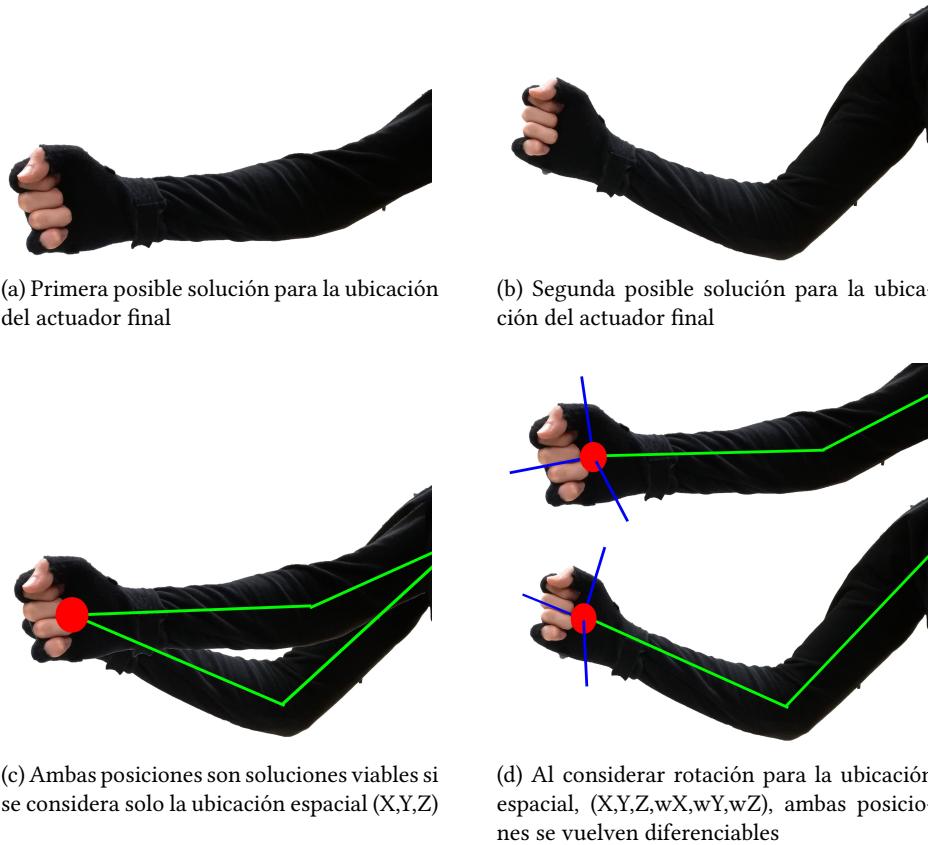


Figura 2.16: Ejemplo de múltiples soluciones para una misma ubicación espacial del actuador final de una cadena de acción



(a) Marcadores de la cabeza



(b) Marcadores de las manos



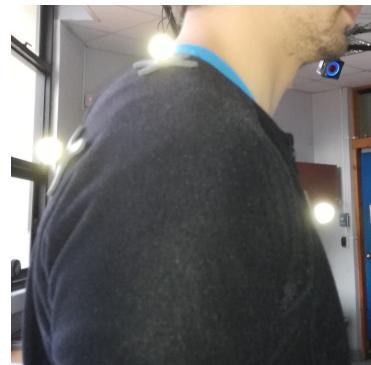
(c) Marcadores de los pies visto de frente



(d) Marcadores de los pies visto de lado



(e) Marcadores del torso visto por detrás



(f) Marcadores del torso visto de lado

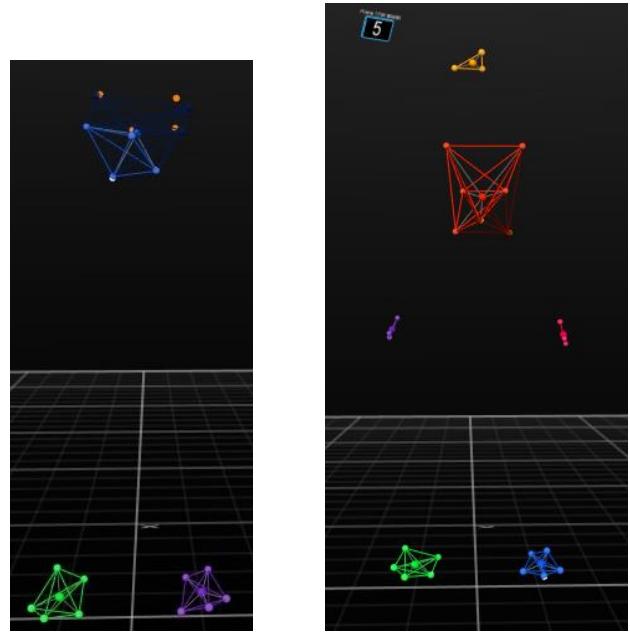
Figura 2.17: Colocación de los marcadores del modelo con un cuerpo rígido por actuador final de las cadenas de acción del NAO, usando un total de 27 marcadores.

Los marcadores que conforman el cuerpo rígido generarán un punto de COM en Motive correspondiente al objeto creado, del cual se exporta la información de ubicación espacial incluyendo su rotación. De este modo, dada la rigidez de los elementos que conforman la cadena y la dependencia de la ubicación de las partes a las que se une cada elemento de la cadena, las soluciones encontradas por el NAO serán menores y con un acomodo en las articulaciones de la cadena que se parezca más al acomodo en la persona que efectuó el movimiento.

Este modelo un poco más complejo con 27 marcadores requiere del etiquetado manual de los objetos de control previo a las grabaciones en Motive para poder identificarlos en el archivo CSV, al igual que el modelo de 6 marcadores, ya que visualmente es posible identificarlos desde Motive (ver Figura 2.18b), pero para el código se requiere de identificadores. Para evitar el etiquetado manual se puede entonces complicar el modelo en Motive utilizando un esqueleto humano con más de 27 marcadores. Esto sería por medio de alguno de los modelos predeterminados de la base de datos de Motive y por tanto todos sus huesos ya estarían etiquetados y diferenciables entre sí por el software de captura al realizar la grabación (Figura 2.10). De estos se puede exportar también la ubicación de los COM de cada hueso, que sería utilizada como el punto de referencia de movimiento para las cadenas de actuadores correspondientes en el robot humanoide NAO. Para usarlo se debe considerar el formato del archivo CSV (que incluiría todos los huesos del modelo esquelético) y hay que encontrar la asignación correcta del hueso del modelo con el actuador respectivo, ya que habrá elementos de sobra al considerar solo las 6 cadenas de actuadores del NAO.

Sin embargo, dado que para la teleoperación del robot NAO se utilizan datos X, Y, Z, wX, wY y wZ para cada cadena de actuador del robot, basta con tener estos datos para que se pueda teleoperar el robot, sin importar tanto el modelo del cuál se obtuvo la información, solo se tiene que asegurar que haya real correspondencia entre la cadena de actuadores con el punto de referencia del cuál se capturó la información para que el movimiento final tenga sentido.

Es importante considerar que dependiendo del movimiento y del modelo usado es más fácil que se pierdan los marcadores y por tanto la figura del modelo utilizado, esto ocurre con mayor frecuencia al utilizar cuerpos rígidos y modelos esqueléticos, sin embargo, por la cantidad de marcadores usados para la creación de cada objeto, si se pierden algunos marcadores aún se puede mantener el cuerpo rígido a lo largo de las tomas, por ejemplo en la Figura 2.18a se observa que aún con varios marcadores perdidos del cuerpo rígido del Tórax (los marcadores naranja corresponden a marcadores no asignados), el objeto sigue siendo identificado como tal debido al resto de marcadores que incluye su definición. Si, en cambio, sucediera esto con un cuerpo rígido definido por el mínimo de marcadores, como las manos con solo 3, entonces se perdería el rastreo del cuerpo en su totalidad.



(a) Ejemplo de cuerpo rígido del Tórax (azul) con marcadores perdidos
 (b) Modelo de cuerpos rígidos completos

Figura 2.18: Modelo de cuerpos rígidos con 27 marcadores totales visto desde Motive, sin marcadores perdidos.

2.3.1. Comparación del cuerpo humano con el del robot humanoide NAO

En general se entenderá que el cuerpo del operador del sistema de teleoperación es más grande que el del robot NAO, y por tanto los datos obtenidos por el MoCap deben ser escalados para corresponder al tamaño del robot (ver Figura 2.15b), además de que por la flexibilidad del ser humano con respecto a la rigidez del robot habrá posiciones que el robot no será capaz de efectuar, como doblar la muñeca. También se debe considerar el posicionamiento de los marcadores para el rastreo óptico sobre el individuo, ya que estos deben colocarse en el exterior del cuerpo humano y además sobre el traje de tela, estos puntos en el espacio deben corresponder al punto de acción que se desea controlar en el robot, es decir, los centros de masa de los actuadores finales de las cadenas de actuadores (en el caso de los modelos de cuerpo rígido y esqueleto humano deben calzar los COM de los objetos rastreados), cuya ubicación es dentro del cuerpo del objeto. Por tanto, además del escalamiento que se debe hacer debido a la diferencia de tamaños entre los cuerpos hay que agregar un pequeño margen de error debido a la colocación de los marcadores con respecto a los COM de los actuadores (ver Figura 2.19) [10].

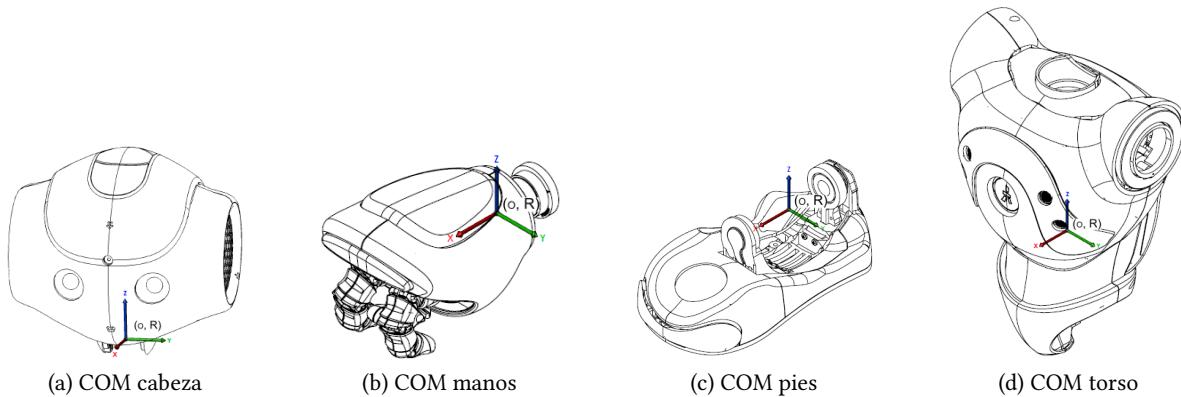


Figura 2.19: Centros de masa de los 6 actuadores finales de las cadenas de actuadores a controlar del robot humanoide NAO, identificados como un marco de referencia de 3 ejes X, Y, Z.

Extraído de: http://doc.aldebaran.com/2-1/family/robots/masses_robot.html

Bajo estas proposiciones es posible encontrar una relación polinomial de grado 1 entre los datos del humano con los del NAO que se conforme de un factor de escalamiento y un *offset* de colocación, como se observa en la ecuación 2.1.

$$\mathbf{y} = \mathbf{x}\mathbf{m} + \mathbf{b} \quad (2.1)$$

Donde \mathbf{y} corresponde al conjunto de coordenadas (X, Y, Z, wX, wY, wZ) que se quiere pasar al robot NAO para la colocación de sus actuadores en el espacio, \mathbf{x} similar a \mathbf{y} pero para los datos de ubicación espacial obtenidos desde el MoCap. Los valores de \mathbf{m} y \mathbf{b} son el conjunto de factores de escalamiento y offset de colocación para las respectivas coordenadas de \mathbf{x} . La ecuación 2.1 debería ser aplicada a cada actuador para realizar el ajuste en el set de coordenadas cartesianas, donde cada uno de los elementos del conjunto de coordenadas (X, Y, Z, wX, wY, wZ) tiene un coeficiente de escalamiento y un offset de colocación.

En primera instancia se realizó el ajuste de manera manual (Figura 2.20) sobre un set de datos correspondiente al movimiento de elevación de los brazos, corroborando el resultado de manera gráfica contra los datos de las posiciones de los actuadores extraídos de los sensores del NAO, y de manera visual comparando el movimiento ejecutado por el NAO con el de la persona para la grabación. Sin embargo, realizar el proceso para diferentes tomas resulta tedioso y no es escalable ya que aplica de manera única para una sola toma y no se garantiza un buen acople con otras grabaciones. Para ello se debe utilizar métodos automatizados que permitan el ajuste para las grabaciones que se quiera, y que además vaya a servir para un proceso de calibración en que se obtenga una relación aplicable a cualquier grabación que se vaya a hacer con la misma persona (si se cambia de sujeto se recalibra y obtiene la nueva relación aplicable a este otro individuo).

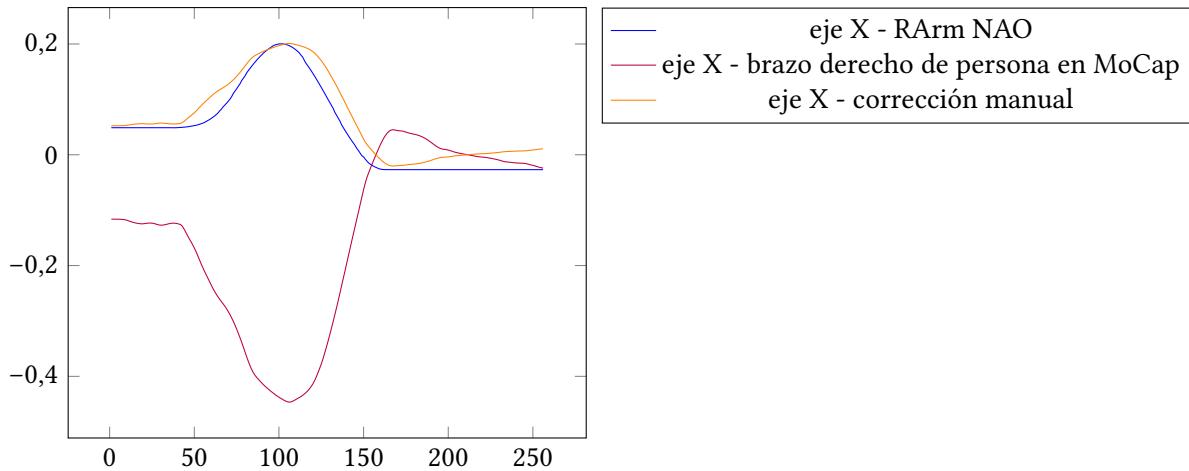


Figura 2.20: Resultado del ajuste de grado 1 manual realizado a los datos del eje X para el movimiento de elevación y retorno a posición inicial de la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 marcadores.

2.3.2. Regresión lineal: Generalidades

Una manera de automatizar el proceso de ajuste es por medio del uso de algoritmos de ajuste de curvas, en este caso de regresión lineal con expresiones polinomiales [16, 35]. El propósito de la regresión lineal es encontrar una relación matemática para un set de valores dependientes (y) con un set de valores independientes (x) por medio de un modelo de curva de grado n que mejor se ajuste a la respuesta del set de datos completo.

Para el caso del ajuste de grado 1 propuesto en la ecuación 2.1, el algoritmo toma en consideración la existencia de un error aleatorio, generando una nueva ecuación (ecuación 2.2) que considera variación en la respuesta. Sin embargo, se trata siempre de buscar que ese error sea el menor posible hasta ser despreciable (≈ 0).

$$y = \beta_0 + \beta_1 x + \epsilon \quad (2.2)$$

Uno de los métodos para resolver el problema de la regresión lineal es el de los mínimos cuadrados [34]. Este método trata de encontrar la curva de mejor ajuste al set de datos minimizando la suma del cuadrado de los residuos⁸ (ecuación 2.3).

$$S = \sum_{i=1}^k (y_i - f(x_i, \beta))^2 \quad (2.3)$$

Donde k es la cantidad de datos que conforman el set (los sets del NAO y del MoCap deben tener la misma cantidad de elementos para el cálculo), y y x son los sets para coordenadas (X, Y, Z, wX, wY, wZ) del Nao y del MoCap respectivamente, y $f(x_i, \beta) = \beta_0 + \beta_1 x$, la predicción del ajuste esperado, es decir,

⁸Diferencia entre la predicción (punto de la curva ajustada generada) y el punto original

los nuevos y que pueden formar la curva ajustada. Estos residuos generan un espacio de soluciones para diferentes valores que pueden tener los coeficientes que se están buscando, el método de mínimos cuadrados minimiza este espacio de soluciones por cada respectivo coeficiente, y el valor del coeficiente en dicho mínimo se entiende por el que permite el mejor ajuste posible para la curva, según los sets de datos.

Hay que tener precaución al realizar estos ajustes a los datos obtenidos por las grabaciones del MoCap y la toma de datos de los sensores del NAO, ya que estos puntos son tomados en el tiempo, y por tanto deben ser comparables para un mismo momento. Aquí hay una ventaja de tener los datos del MoCap en una grabación por sobre la transmisión en tiempo real, ya que se puede hacer la corrección temporal y procesar los datos hasta tener la seguridad de que los datos concuerdan, sin correr riesgos de un mal ajuste inevitable al tener que enviar los datos en tiempo real. Este ajuste puede ser manual, una persona se encarga de graficar los datos y escoger los rangos que hagan que sean comparables en el tiempo, o bien se podría automatizar con la implementación de un módulo de detección de patrones que acomode los datos del MoCap según la *forma* con respecto a los del NAO, para luego aplicar el ajuste por regresión.

Al graficar el comportamiento de los datos del NAO contra los datos del MoCap para corroborar la correcta elección de la forma descrita por la ecuación 2.1, se obtiene la Figura 2.21 que apunta más a una relación polinomial de grado 2.

Se puede observar las dos curva con aparente forma cuadrática para un mismo movimiento del brazo derecho en el eje X, una correspondiendo a la elevación del brazo (partiendo con el brazo extendido en dirección al suelo), y la otra curva representando el movimiento en reversa. Otro ejemplo es la Figura 2.22, que corresponde al eje Z de la misma grabación para la misma extremidad.

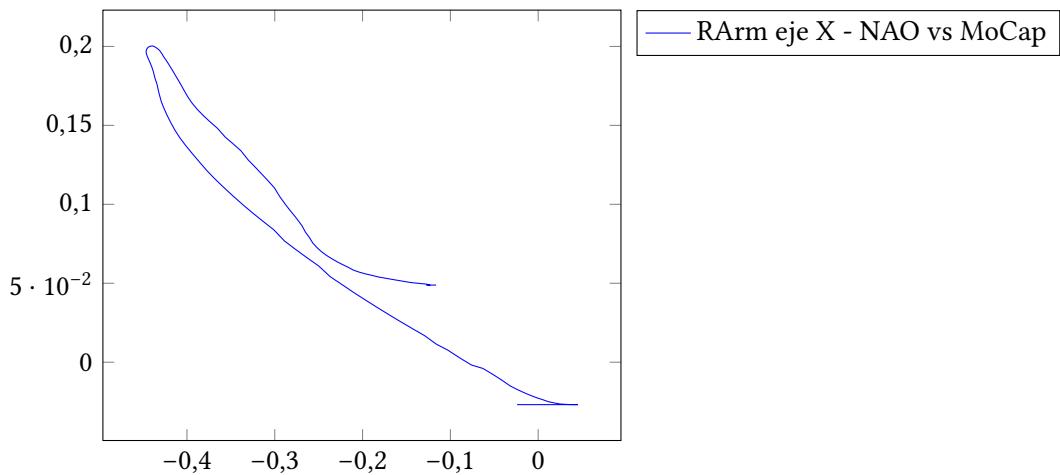


Figura 2.21: Relación entre datos del NAO vs los del MoCap para la cadena RArm, sobre el eje X. Grabación de 10 s subiendo y bajando los brazos. Modelo de 6 marcadores. Eje X del MoCap apunta en dirección opuesta al del NAO.

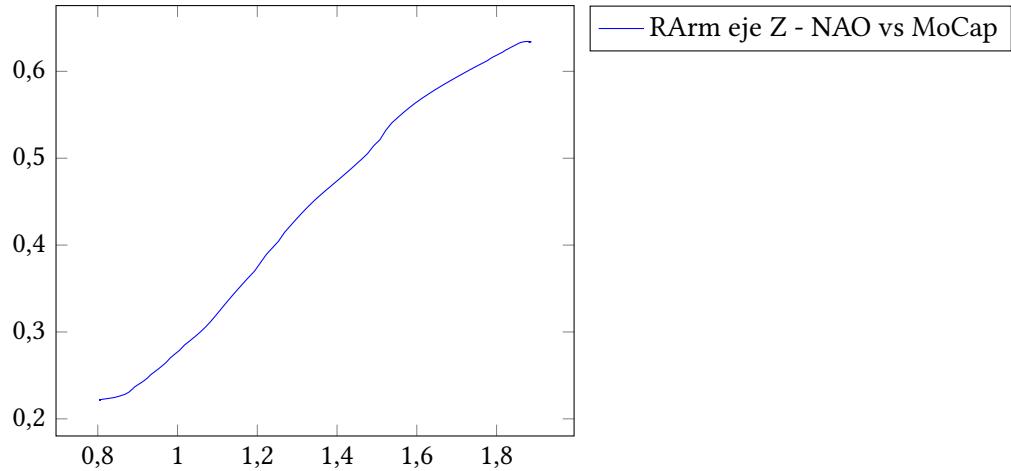


Figura 2.22: Relación entre datos del NAO vs los del MoCap para la cadena RArm, sobre el eje Z. Grabación de 10 s subiendo y bajando los brazos. Modelo de 6 marcadores

Dada esta sugerencia visual se procede a realizar ajustes con regresión lineal de grado 2 para los datos, y en vez de la ecuación 2.1 se obtiene entonces la forma de la ecuación 2.4.

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 \quad (2.4)$$

Este algoritmo permite el ajuste de la Figura 2.23 de manera automática. Se puede observar un pequeño desplazamiento entre los puntos máximos de las curvas del NAO y los datos con corrección debido a diferencias de ubicación temporal entre los datos. Estas diferencias en los tiempos y cuadros de las tomas con respecto a la grabación de calibración del NAO afecta no solo en cómo se observan los datos al compararlos en gráficas, sino en sí en la obtención de los coeficientes del ajuste de la regresión al no comparar los datos independientes (x) con sus verdaderos respectivos dependientes (y) en el tiempo. Por tanto, es realmente importante que al editar las grabaciones del MoCap se tenga bien presente los cuadros que deben formar parte del set de datos para poder hacer la mejor comparación posible. Inclusive la implementación de algún módulo que permita realizar este recorte en los sets de datos para no depender del buen ojo de una persona.

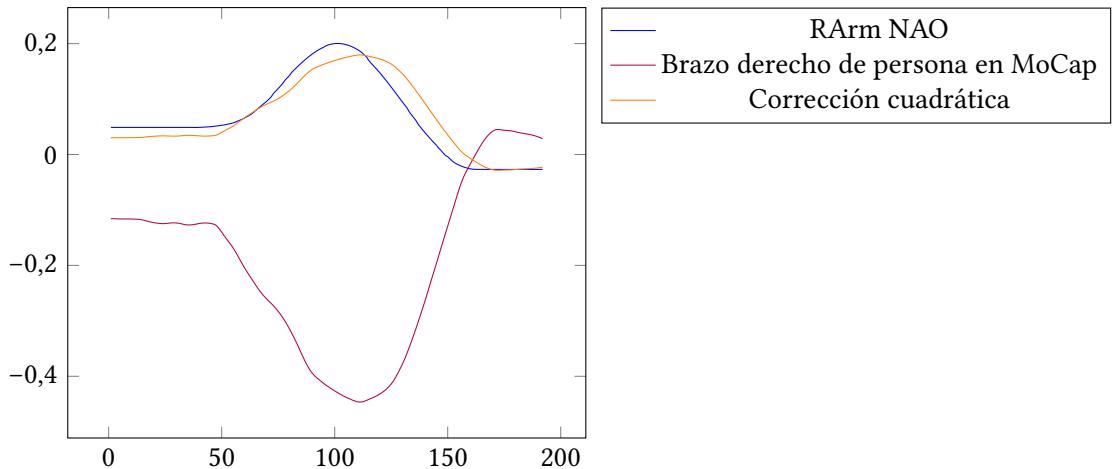


Figura 2.23: Resultado del ajuste polinomial de grado 2 realizado a los datos del eje X para el movimiento de elevación y retorno a posición inicial de la cadena de actuadores RArm del robot humanoide NAO. Marco de referencia ROBOT. Modelo de 6 marcadores. Regresión por función *polyfit* de Python.

Cuando la curva se muestra como una recta horizontal (o lo más parecido) representa que el punto en estudio mantiene su posición *constante* durante cierto tiempo sobre el eje en estudio. Estos casos no ayudan mucho a la toma de datos ya que no hay información para realizar el ajuste correcto, por ello se debe asegurar de mover cada actuador en todos sus DoF como parte del proceso de calibración. Casos similares de mal ajuste se pueden dar si los datos no concuerdan temporalmente, o si se comparan datos incorrectos (por ejemplo mezclar datos del brazo con una pierna), a como se observa en la Figura 2.24.

El problema observado en la Figura 2.24 se debe a un fallo de calibración por enviar datos de una pierna para obtener la regresión de los datos de un brazo, allí la importancia de tener bien identificados qué datos pertenecen a qué actuador. Además, para el evento ilustrado en la misma imagen los brazos realizan un desplazamiento notable en los ejes X (van al frente y regresan cerca de la posición 0 en X) y Z (suben y bajan), pero para la coordenada en Y no se da ningún cambio ya que el movimiento ocurre sobre este eje, de modo que no hay información sobre cómo se comporta este eje. Es necesario aplicar un proceso de calibración que considere estos casos y los actuadores involucrados para adquirir la mayor cantidad de información útil posible, es decir, que haya movimiento sobre todos los DoF a usar.

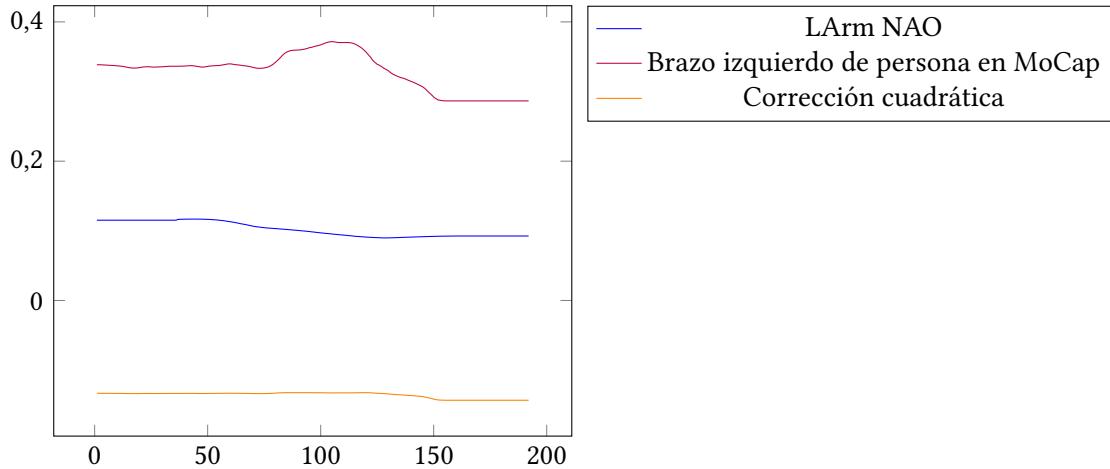


Figura 2.24: Resultado del ajuste de grado 2 realizado a los datos de coordenada Y para el movimiento de elevación y retorno a posición inicial de la cadena de actuadores LArm del robot humanoide NAO con datos provenientes de otra cadena. Marco de referencia ROBOT. Modelo de 6 marcadores. Regresión por función *polyfit* de Python.

En el capítulo 4 y el apéndice A se muestran resultados de usar tanto regresión lineal de grado 1 como de grado 2.

2.3.3. Diseño de un proceso de calibración

La intención de la existencia de un proceso de calibración es permitir que cualquier persona pueda fungir como operador en el sistema de teleoperación, obteniendo una ecuación de ajuste a sus datos que pueda ser guardada para futuros usos, y que sea aplicable a cualquier grabación que haga esa persona para mover al NAO.

Lo primero que se ocupa es tener una serie de rutinas de movimiento para obtener los datos del NAO. Las rutinas se describen en el Apéndice B. Considerando que las rutinas permiten la obtención de suficiente información para cada coordenada de cada actuador, estos datos deben ser almacenados para ser usados como punto de comparación en cada proceso de calibración, sin ser modificados a menos que se vaya a mejorar la colecta.

Para generar las rutinas necesarias se utiliza el modo Animación de Choregraph. Se trata que cada movimiento sea de no más de 12 s y enfocados a generar variación en los datos de los 6 DoF de un solo actuador o un set de actuadores. Por ejemplo, RArm y LArm se pueden obtener de una sola animación al mover los brazos en espejo. Para el caso de las piernas (RLeg y LLeg) es mejor tener una animación independiente para cada una ya que el balance del NAO cobra mayor importancia porque se deberá elevar un pie en algún momento para poder abarcar los tres ejes cartesianos. El torso y la cabeza (Torso y Head) se manejan independientes también. El sistema de teleoperación incluye un módulo que permite la extracción de la ubicación espacial de los actuadores desde el NAO.

Obtenidos los datos del NAO, se requiere tener los de la persona a comparar. Para esto el individuo deberá imitar los movimientos de las rutinas ejecutadas por el NAO, tratando de seguir el ritmo de la acción. Esto ya que se ocupan los datos correspondientes para realizar la regresión lineal y que estos concuerden temporalmente para que el ajuste sea acertado y se facilite la verificación visual, se deben sincronizar bien los movimientos. En este caso es recomendable que la persona ejecute los movimientos guiándose con el robot humanoide NAO en acción en tiempo real (se debe tomar las previsiones al caso de la interferencia en el MoCap por las luces LED's del robot, ver Figura 2.7b).

En este proceso es importante la manera en que la persona ejecuta la rutina, en especial si se están utilizando cuerpos rígidos para obtener rotaciones ya que es fácil que se pierda algún marcador y por tanto la figura del objeto delimitado por el cuerpo rígido desaparecerá junto con la información del mismo por los cuadros de tiempo en que sucedió el evento. Se debe asegurar que la persona esté posicionada donde todas las cámaras del MoCap perciban todos los marcadores para disminuir este riesgo, también los movimientos de la persona deben ser tales que las restricciones que permiten la creación de un cuerpo rígido no sean violadas. Esto sucede, por ejemplo, con los marcadores colocados en los brazos que, debido al movimiento de giro que puede realizar el antebrazo, es fácil violar estas restricciones no intencionalmente con movimientos que son naturales para una persona. Al utilizar el modelo de 27 marcadores esto ocurre con las manos al abrir y cerrarlas, dependiendo de cómo se hayan colocado los marcadores respectivos.

Además, es importante recordar que la configuración de los marcadores que delimitan cada área de control debe variar entre cada cuerpo rígido para prevenir que el sistema de captura los confunda en la ejecución de la grabación.

Una vez que se tienen los datos del NAO y de la persona, se debe realizar el ajuste entre los sets de cada actuador para las coordenadas (X, Y, Z, wX, wY, wZ). Para ello se utiliza el algoritmo de regresión lineal con mínimos cuadrados (en este caso, para futuros proyectos se puede hacer pruebas con otros métodos de ajuste de curvas). El módulo de calibración retornará al usuario los coeficientes del ajuste para las grabaciones generales realizadas por la persona.

Capítulo 3

Implementación

El capítulo consiste en la descripción de los diferentes módulos funcionales que conforman el sistema de teleoperación, mostrados con respecto a los diferentes pasos que se requiere para que el sistema funcione según los requerimientos planteados en el capítulo 2. El código completo se puede observar en el apéndice C, o en el repositorio Git del proyecto: https://github.com/LennonNM/Proyecto_Electrico.

El sistema de teleoperación presente fue desarrollado en Python¹ 2.7.14, la razón de escogencia de este lenguaje de programación proviene de antecedentes en proyectos desarrollados en CORE para el PRIS-Lab, por ejemplo el proyecto NAO Remote con el joystick, en que el uso de Python permitió realizar pruebas a la implementación en una temprana fase del desarrollo del proyecto debido a la comodidad para generar un script con una prueba corta, y utilizar módulos de Python como Pygame que facilitó el uso del joystick. Sin embargo, este sistema de teleoperación podría ser implementado en otros lenguajes como C++, siguiendo las ideas y modelos propuestos con anterioridad en este documento (o bien, escogiendo otro camino de implementación al que se propuso).

Se usa el modelo de 6 marcadores en el desarrollo inicial del proyecto para definir diferentes requerimientos de operación y diseño del sistema. Se realizó ajustes al código antes de su versión final para incluir el funcionamiento con datos de rotación provenientes de un modelo 3D con un cuerpo rígido por actuador final de la cadena de acción del NAO, utilizando 27 marcadores (ver Figura 2.17), procesando un archivo CSV con el formato de exportación para cuerpos rígidos de Motive. El sistema podría usarse también con datos provenientes de un modelo 3D de esqueleto humano con N marcadores (como el propuesto de 42 marcadores de la Figura 2.10), sin embargo el archivo CSV a procesar deberá ser modificado para que calce con el formato de cuerpo rígido, dado que no se ha incluido la lectura del formato de cuerpo esquelético en esta implementación (o bien, modificar el código para que se ajuste al formato de exportación CSV para cuerpos esqueléticos de Motive). En cuanto al ajuste de curvas se puede utilizar una regresión lineal con polinomio de grado 1 y 2 para obtener los coeficientes de calibración, en el capítulo 4 y el apéndice A se comparan los resultados de usar cada grado; además, para la ejecución del movimiento del robot NAO se permite la libertad de escoger si se quiere ubicación espacial (X,Y,Z) o ubicación con rotación (X,Y,Z,wX,wY,wZ).

El enfoque del movimiento del robot humanoide NAO es de la colocación de los brazos en el espacio, sin embargo, se incluye en el código secciones que permiten controlar las piernas y el torso (de-

¹"Using Python is one of the easiest ways to program with NAO", <http://doc.aldebaran.com/1-14/dev/python/index.html>

pendiendo del marco de referencia utilizado), esto para sentar una base para iniciar futuros proyectos de continuación y mejora del sistema de teleoperación propuesto.

3.1. Operaciones necesarias que debe cumplir el sistema de teleoperación

Según lo descrito en el capítulo 2, el sistema de operación ha de satisfacer con los siguientes requisitos en funcionamiento:

1. Calibración del sistema de teleoperación
 - a) Rutinas de calibración guardadas para el NAO (archivo CSV)
 - b) Datos de la persona imitando las rutinas de calibración del NAO
 - c) Lectura de sets de datos (archivos CSV)
 - d) Regresión lineal en sets de datos
 - e) Almacenar coeficientes del ajuste humano-NAO (archivo CSV)
2. Obtención de la grabación a imitar
 - a) Grabación de coreografía de interés por MoCap (Motive)
 - b) Exportar datos en un formato conocido (archivo CSV)
3. Ajuste de datos de la grabación
 - a) Lectura de datos del movimiento humano (archivo CSV)
 - b) Ajuste a los set de datos utilizando los coeficientes obtenidos de la calibración
4. Generación de instrucciones para el control del robot humanoide NAO
 - a) Obtención de datos de la grabación corregidos
 - b) Acomodar la información necesaria según el formato de las diferentes funciones a usar para el control de cuerpo completo del NAO
5. Ejecución de la imitación por parte del NAO
 - a) Posicionar al NAO de manera segura para iniciar con el control del movimiento
 - b) Manejo del movimiento de cuerpo completo del NAO por medio de las instrucciones generados tomando en cuenta balance y peligro de caídas
 - c) Posicionar al NAO de manera segura al finalizar la coreografía

Según el propósito de la rutina a ejecutar (solo posicionamiento o movimiento completo) los archivos CSV obtenidos de Motive incluirán solo coordenadas (X,Y,Z) o además tendrán rotación espacial (wX,wY,wZ). Esto debe concordar con el modelo utilizado para la colocación de marcadores y con la calibración realizada, si no se incluye rotación solo se podrá calibrar X, Y y Z para cada actuador y se considerará los datos de rotación con valor 0. Se dará por entendido que si se usa ubicación sin rotación el modelo utilizado para el MoCap será el de 6 marcadores flotantes, y si se incluye rotación corresponderá al de cuerpos rígidos con 27 marcadores, y por tanto se espera el formato de exportación CSV respectivo a cada modelo.

Para cumplir con las tareas recién descritas se divide el código en dos conjuntos, el primero son todos aquellos archivos cuyo contenido es la definición de funciones que cumplen tareas detalladas y específicas, estos llevan la terminación *_Func* en su nombre; el segundo conjunto son aquellos cuyo propósito es cumplir con algún proceso conformado por las tareas específicas que realizan las funciones que incluyen en su contenido, estos simplemente **no** tienen la terminación *_Func* (ver Figura 3.1), en general estos últimos tratan de ejecutar los procesos de calibración (*Calibrate.py*), obtención de datos directamente del robot NAO (*GetPositions.py*) y la ejecución del movimiento del robot humanoide NAO como imitador de la grabación (*Move.py*); la interacción de estos dos conjuntos permite el cumplimiento de las tareas requeridas en diferentes módulos de operación (módulo de calibración, módulo de ejecución de movimiento) y permiten simplificar el mejoramiento del sistema actual hacia una versión más robusta y compleja al solo tener que añadir módulos que cumplan con esas tareas deseadas, considerando algunos cambios en los módulos ya implementados.

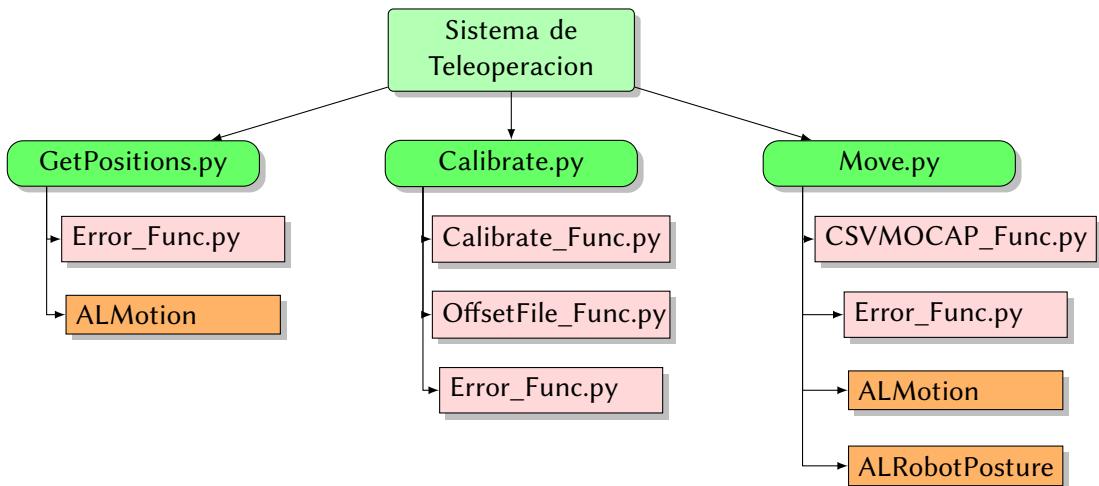


Figura 3.1: Estructura del código del sistema de teleoperación

Las siguientes secciones del capítulo describen las tareas específicas que se deben realizar para satisfacer con el proceso de teleoperación, siendo estas la capacidad de leer y escribir archivos CSV (sección 3.2), obtención y generación de datos de posicionamiento espacial desde el NAO y el MoCap (sección 3.3), la calibración del sistema de teleoperación por regresión lineal (sección 3.4), el ajuste de los datos de una grabación por medio de las relaciones obtenidas de la calibración (sección 3.5) y la

ejecución del movimiento del NAO como imitador de la grabación (sección 3.6). La sección 3.7 resume las descripciones de las secciones anteriores en una pequeña guía de uso del sistema de teleoperación.

3.2. Lectura y escritura de archivos CSV

EL proyecto actual utiliza el formato CSV para almacenar los diferentes sets de datos a utilizar en la teleoperación. Estos datos pueden ser:

- Coeficientes de la regresión lineal
- Datos de posicionamiento espacial de las rutinas de calibración del NAO
- Datos de posicionamiento espacial del humano imitando las rutinas de calibración del NAO
- Datos de posicionamiento espacial de la grabación que se quiere imitar

Para cada una de estas posibilidades se requiere un archivo independiente para contener la información (en el caso de la calibración se requeriría uno por rutina de calibración según el apéndice B), estos con sus propios formatos. Recordar que el formato CSV, visto como filas y columnas, separa los elementos de una misma fila con una coma (,) y se termina e inicia una nueva fila con cambio de línea. Los espacios en blanco son considerados como parte del elemento mientras esté delimitado por la coma, esto implica que se puede tener un elemento vacío al escribir „; si no se quiere incluir datos a una posición (fila, columna), se deberá dejar el espacio en blanco pero incluyendo el delimitador de coma para indicar que es una casilla con valor vacío (ejemplo: dato1, , dato3). Es de suma importancia que los datos estén organizados y ubicados correctamente para que la lectura del archivo sea correcta y no haya confusión con lo que se esperaba leer, o bien la ausencia de datos necesarios para la calibración.

El formato para el archivo que almacenará los coeficientes de calibración es el siguiente:

```
D#,rot
B2,B1,B0
```

Donde **D#** corresponde al grado del polinomio usado para la regresión lineal, **rot** indica si incluye rotaciones (yes) o no (no), y **B2**, **B1** y **B0** son los coeficientes de la relación polinomial de la ecuación 2.4, o bien serán **B1** y **B0** si se usa polinomio de grado 1. Se considera por tanto que la primera fila es el encabezado del archivo, y las filas siguientes tendrán los datos del ajuste, un DoF por fila siguiendo el orden X, Y, Z (wX, xY y wZ si se da el caso), estas filas por cada cadena de actuadores siguiendo el orden preferente **RArm**, **RLeg**, **LLeg**, **LArm**, **Torso** y **Head**, es decir, tres filas (ó 6 si incluye rotación) por cada actuador final a controlar.

El formato de los archivos con datos exportados del MoCap cumplen la estructura definida por OptiTrack para la exportación de archivos CSV desde Motive [37] incluyendo el encabezado con información descriptiva de los datos (ver Figura 3.2).

```

Format Version,#,Take Name,Name,Capture Frame Rate,#,Export Frame Rate,#,
Capture Start Time,###,Total Frames in Take,#,Total Exported Frames,#,
Rotation Type,Type,Length Units,Units,Coordinate Space,Space Name
,,,,,,,,,,,
,,Marker,Marker,Marker
,,Name,Name,Name
,,ID#,ID#,ID#
,,Position,Position,Position
Frame,Time,X,Y,Z
#,#,#,#,#,#,#

```

O bien, si incluye rotaciones como el caso de la exportación de datos de cuerpos rígidos se agregan más columnas por *Marker*, siendo este identificador el del marcador (modelo de marcadores) o del COM del objeto (modelo cuerpos rígidos). Nótese que en el caso de las rotaciones estas aparecen antes que las posición X,Y,Z, hay considerar este cambio al tratar ya sea el modelo de 6 marcadores o el de cuerpos rígidos con rotaciones. También se incluye el error por marcador, este valor es un error de cálculo incluido ya que el COM del cuerpo rígido es una aproximación según la ubicación de cada marcador físico que conforma el objeto. En esta implementación no se utiliza pero se consideró incluirlo como parte de los datos exportados para futuros cambios del sistema en que resulte útil este dato

```

Format Version,#,Take Name,Name,Capture Frame Rate,#,Export Frame Rate,#,
Capture Start Time,###,Total Frames in Take,#,Total Exported Frames,#,
Rotation Type,Type,Length Units,Units,Coordinate Space,Space Name
,,,,,,,,,,,
,,Rigid Body,Rigid Body,Rigid Body,Rigid Body,Rigid Body,Rigid Body,
Rigid Body
,,Name,Name,Name,Name,Name,Name
,,ID#,ID#,ID#,ID#,ID#,ID#
,,Rotation,Rotation,Rotation,Rotation,Position,Position,Position,
Error Per Marker
Frame,Time,X,Y,Z,W,X,Y,Z,
#,#,#,#,#,#,#,#,#

```

Format Ver	1.21	Take Name	back roll3	Capture Frame	120	Export Frame	120	Capture Si	2014-12-0	Total Fran	
Name		Bone	Bone	Bone	Bone	Bone	Bone	Bone	Bone Mar	Bone Mar	Bon
		SkeletonA_H	SkeletonA_H	SkeletonA_Hip	SkeletonA_Hip	SkeletonA_H	SkeletonA_H	SkeletonA_H	SkeletonA_Skel	SkeletonA_Skel	SkeletonA_Skel
		1	1	1	1	1	1	1	10005	10005	
ame	Time	Rotation	Rotation	Rotation	Rotation	Position	Position	Position	Position	Position	Posi
	X	Y	Z		W	X	Y	Z	X	Y	Z
0	0	0.037632	-0.021128		0.003814	-0.999061	1.31038	0.930015	1.346092	1.462953	1.032686 1.4
1	0.008333	0.037632	-0.021128		0.003814	-0.999061	1.31038	0.930015	1.346092	1.462953	1.032686 1.4
2	0.016667	0.034949	-0.021491		0.003548	-0.999152	1.311624	0.930398	1.345122	1.464207	1.032758 1.4

Figura 3.2: Ejemplo de visualización en un LibreOffice Calc de un archivo con datos de cuerpos rígidos exportado a CSV desde Motive

Las primeras 7 filas (iniciando cuenta en 1) corresponden al encabezado (se debe marcar la opción de incluir encabezado al exportar el archivo desde Motive). La información más importante de esta sección se encuentra en las filas 6 y 7, en la 6 se indica la etiqueta del marcador, o cuerpo rígido, al que corresponde cada columna de datos de coordenadas (importante haber etiquetado correctamente cada marcador/cuerpo rígido), si hay más de 6 etiquetas diferentes (aparecerá una columna con la etiqueta por cada DoF incluido) se deben ignorar esas columnas extra ya que no corresponden a los únicos 6 actuadores finales definidos a controlar **RArm**, **RLeg**, **LLeg**, **LArm**, **Torso**, **Head**. En la fila 7 cada elemento es el indicador de qué son los datos de cada columna. La columna del número de cuadro se puede ignorar ya que esta información no es utilizada, la de tiempos y datos de cada coordenada se utilizan para el control, de modo que es importante recolectar los datos y agruparlos en los sets correctos, es decir, que no se mezcle la información. A partir de la fila 8 se incluyen los valores numéricos correspondientes a las coordenadas.

Para este archivo es importante el etiquetado de los puntos de control correspondientes a los actuadores del NAO para no confundir datos. Si se usa el etiquetado manual (al usar el modelo de cuerpo rígido o de 6 marcadores) se deben usar los nombres de las cadenas de actuadores para hacer la relación directa. En el caso de haber utilizado el modelo de esqueleto humano, el etiquetado se realiza según los nombres designados por el modelo de manera automática por Motive, al leer el archivo se debe hacer la relación con los nombres de los actuadores para saber a qué corresponde cada columna de datos.

Para los archivos que contienen los datos extraídos desde el NAO se ha decidido acomodarlos siguiendo el formato de espaciado de los archivos exportados desde Motive, con la variación de que las primeras 6 filas son espacios en blanco (que de igual manera se ignoran), la séptima fila denota qué dato contienen cada columna para guiar al lector, ya que de igual manera se ignora en la lectura y se asume que los datos son acomodados con el orden preferente de los actuadores **RArm**, **RLeg**, **LLeg**, **LArm**, **Torso** y **Head**, con el orden de las coordenadas (X, Y, Z, wX, wY, wZ), con la opción de no incluir rotaciones.

Para satisfacer estos requerimientos en formatos y archivos necesarios se utilizan las funciones descritas en **CSVMOCAP_Func.py**, **OffsetFile_Func.py** y **GetPositions.py**. Esto lo hacen por medio de diferentes operaciones disponibles en las librerías de python *time*, *csv*, *itertools* y *os*. En general el proceso de escritura y lectura se basa en extraer los datos completos y almacenarlos en una lista, esta lista es editada y recorrida en todos sus elementos como filas y columnas, según la representación visual del CSV, eliminando la información que no se va a requerir, leyendo la que sirve como guía, y

extrayendo y almacenando los datos útiles para el movimiento del robot NAO.

GetPositions.py se encarga de escribir un archivo CSV con los datos de posicionamiento espacial obtenidos de la lectura de los sensores del NAO, acomodando los datos según el formato descrito previamente. Incluye las columnas de rotación si se le solicita, ya que se debe saber la existencia del uso de las rotaciones para la lectura de este archivo. Se utiliza la fecha y hora de ejecución del código como parte del nombre para generar archivos únicos que no sean sobre-escritos cada vez que se corre el código, o bien, el usuario puede seleccionar un nombre para el archivo.

OffsetFile_Func.py contiene funciones de lectura y escritura del archivo CSV con los coeficientes de la regresión lineal.

CSVMOCAP_Func.py incluye funciones de lectura del archivo CSV con datos del MoCap exportados desde Motive, y permite obtener los vectores de tiempo, coordenadas ya corregidas y actuadores a utilizar según el marco de referencia solicitado, para ser usados en las instrucciones de control de movimiento del NAO.

3.3. Obtención y generación de datos de posición espacial

Los datos de posicionamiento espacial que se ocupan para que el sistema de teleoperación funcione son:

- Rutinas de calibración del NAO
- Rutinas de calibración del operador
- Grabación del operador a imitar por el NAO

Para la grabación a imitar y las rutinas de calibración del operador se utiliza la exportación de datos CSV desde Motive. Para iniciar una grabación se deben seguir los pasos descritos en las guías de uso oficiales del MoCap [42]), al igual que las consideraciones para el proyecto y descripciones del equipo en el capítulo 2. Hay que tomar en cuenta que se deben editar las grabaciones para que estas no contengan cuadros de grabación con datos ausentes, ya que esto generará espacios vacíos en el archivo CSV que desplazarán el orden esperado para la lectura (ocurrirá un error de lectura por encontrar un valor en blanco). En el caso de las rutinas de calibración es importante recordar el largo de la grabación de modo que se incluyan solo los movimientos necesarios que correspondan a las rutinas de comparación del NAO, por ejemplo, al usar un modelo esquelético se suele comenzar las grabaciones con pose T, pero el trayecto de pose T a Stand no se utiliza para la calibración y se debe eliminar de la grabación a usar, tratando que los tiempos de ejecución calcen ya que es importante la relación entre puntos del set de datos del operador con los del NAO en la línea temporal para obtener un buen ajuste en la regresión lineal.

En el caso de los datos de posicionamiento espacial de los actuadores del NAO para las rutinas de calibración se utiliza el script **GetPositions.py**, que hace uso de la instrucción `getPositions()` del API ALMotion del robot NAO. Esta instrucción devuelve una lista para la cadena de actuadores seleccionada con los valores de coordenadas X, Y y Z, y rotación. Para obtener los datos de los sensores del NAO se debe correr **GetPositions.py** mientras se ejecuta alguna de las rutinas de calibración del NAO desde Choregraphe.

3.4. Calibración del sistema de teleoperación

La calibración parte de la comparación entre datos de movimiento del operador con los del NAO, de donde se obtiene una ecuación matemática que transforma las posiciones del MoCap en posiciones equivalentes para los actuadores del NAO. Este proceso debería hacerse una única vez por operador antes de teleoperar al NAO para que imite una grabación. Los coeficientes de ajuste se almacenan en el archivo **offsets.py** por defecto, o bien, el usuario puede especificar la extensión al nombre del archivo para separarlos por operador calibrado.

El proceso utiliza las funciones de **Calibrate_Func.py** para realizar el ajuste de los sets de datos por medio de regresión lineal, utilizando la función `polyfit` de la librería **numpy**, de Python, que calcula los coeficientes según el grado del polinomio indicado, esto se realiza para cada coordenada (pasado como una lista de datos) para una sola cadena de actuadores. La calibración completa se realiza con

Calibrate.py, encargado de las llamadas a las funciones de calibración con los datos requeridos para obtener los coeficientes a utilizar en el ajuste, también utiliza las funciones de **OffsetFile_Func.py** para escribir los valores obtenidos en el archivo de coeficientes indicado.

```

1 #Ajuste con polinomio grado 2 para las curvas (pX, pY, pZ) con datos del operador ↔
   y (naoX, naoY, naoZ) con datos del NAO, para coordenadas X, Y y Z
2 degree = 2
3 p1[0] = list(np.polyfit(pX, naoX, degree))
4 p1[1] = list(np.polyfit(pY, naoY, degree))
5 p1[2] = list(np.polyfit(pZ, naoZ, degree))

```

En el momento de la grabación en MoCap el operador debe asegurarse que los movimientos los ejecute con las velocidades y en los momentos correctos, para ello el NAO debe ejecutar la rutina respectiva para que la persona le siga mientras se toman los datos. Se utilizó el modo animación de Choregraphe para la creación de las rutinas de calibración, de modo que en el momento de la calibración se debe correr la animación desde Choregraphe e iniciar con la grabación en Motive. Todas las rutinas inician con los brazos extendidos en dirección al suelo (pose predeterminada **Stand** del NAO, ver figura 2.12a), por lo que el operador deberá iniciar con la misma pose.

Solo cuando se desea generar los datos de calibración del NAO se deben correr las rutinas de calibración en Choregraphe junto con **GetPositions.py**, el proyecto incluye la captura de estos datos en el directorio .../Calibration/NAO/RigidBody_Defaults, estos incluyen datos de rotación que son ignorados si la calibración se realiza solo para X, Y y Z.

3.5. Ajuste de datos de una grabación del MoCap

Para realizar el ajuste a los datos de la grabación que se desea que el robot humanoide NAO imite se utilizan las funciones de **CSVMOCAP_Func.py**, que se encargan de leer el archivo CSV con las posiciones espaciales de los marcadores rastreados por MoCap, y el archivo *offsets.csv* (o el que se indique) que contiene los coeficientes del ajuste polinomial.

La lectura del archivo CSV se realiza suponiendo que este satisface el formato de exportación de datos desde Motive, incluyendo encabezado. Se realiza la lectura de los tiempos de ejecución de cada cuadro y se almacenan como una lista de tiempos. Se lee la fila con los indicadores de los nombres de los marcadores (actuadores) para saber el orden de aparición de las coordenadas y no mezclar los datos, para tener los datos correctos se debe de haber etiquetado los marcadores (o COM de cada cuerpo rígido) con los nombres **RArm**, **RLeg**, **LLeg**, **LArm**, **Torso** y **Head** según correspondan, ya que no se hace ningún tipo de análisis sobre los datos o comparación con rangos esperados para saber a qué extremidades corresponden los datos. El etiquetado se realiza de manera manual al editar la grabación antes de exportar los datos, para no hacer el etiquetado por cada grabación se recomienda realizarlo antes de comenzar cualquier grabación dentro del mismo proyecto de Motive, de modo que se tendrá que etiquetar solo una vez y para el resto de grabaciones que se realicen ya estarán las etiquetas puestas y listas para la exportación. También se puede etiquetar luego de haber realizado todas las grabaciones, pero se deberá etiquetar cada toma.

Luego se extraen todos los valores de las posiciones espaciales asumiendo que se muestran como X, Y, Z (o bien wX, wY, wZ, X, Y, Z en caso de incluir rotación). Es importante que no hayan espacios vacíos, antes de exportar los datos desde Motive se debe revisar la grabación y corregir los cuadros en que se pierde el rastreo de los marcadores, o bien, manualmente ingresando valores en las casillas del archivo CSV.

Por comodidad de uso se ordenan los diferentes sets de coordenadas en el orden preferente de actuadores **RArm**, **RLeg**, **LLeg**, **LArm**, **Torso** y **Head**, de este modo se puede tener certeza a qué cadena de actuadores corresponde cada lista de posiciones y que siempre estarán en el mismo orden. En el momento del ordenamiento se realiza además el ajuste de los datos, almacenando el resultado de la relación de la ecuación 2.4. En este punto se realizan además otros ajustes que se puedan agregar para corregir situaciones de grabación, por ejemplo, se debe intercambiar el orden de las coordenadas recibidas del MoCap de (X, Y, Z) con (X, Z, Y). Este ajuste es necesario por la diferencia existente en el marco de referencia del robot NAO y del generado en Motive con la escuadra de asignación del suelo, en que el eje X aparece invertido en valores positivos y negativos, y los ejes Y y Z están intercambiados. Esto se observó en las pruebas realizadas al ejecutar el movimiento del NAO, luego de realizar este ajuste la intención del movimiento observado concordaba la intención de la rutina grabada. Esta corrección es utilizada además en las funciones de **Calibrate_Func.py** para que exista concordancia entre los códigos y la aplicación de los coeficientes obtenidos.

Antes de utilizar los datos corregidos es importante además considerar si se va a utilizar el marco de referencia **ROBOT** o **TORSO**, si se utiliza ROBOT los datos están listos, ya que el marco de referencia utilizado para la calibración y la grabación a imitar concuerda con este. Si se quiere utilizar TORSO se debe hacer un ajuste extra, que corresponde a relacionar todas las cadenas de actuadores con respecto a la cadena **Torso**, que coincide con el origen de este marco de referencia. El ajuste se realiza calculando la diferencia entre las coordenadas de ubicación del Torso con las coordenadas de ubicación del actuador que se quiere desplazar al marco de referencia, ver ecuación 3.1. Nótese en el código que se supone que *listaDeActuadores[4]* siempre va a corresponder a los datos del Torso, esto es gracias a la etapa de ordenamiento de datos.

```

1 #RArmTorso = RArmROBOT - TorsoROBOT
2 for i, item in enumerate(listaDeActuadores[0]):
3     for j, item2 in enumerate(listaDeActuadores[0][j]):
4         listaDeActuadores[0][i][j] = round((listaDeActuadores[0][i][j] - ←
5             listaDeActuadores[4][i][j]), 2)

```

$$CA_{Tors} = CA_{ROBOT} - CT_{ROBOT} \quad (3.1)$$

Donde CA_{Tors} corresponde a la ubicación espacial del actuador reubicado con respecto al marco **TORSO**, CA_{ROBOT} es la ubicación espacial del actuador con respecto al marco de referencia inicial **ROBOT**, y CT_{ROBOT} es la ubicación espacial de la cadena **Torso** en el marco de referencia **ROBOT**, coincidente con el marco de referencia deseado **TORSO**.

Además de los marcos de referencia **ROBOT** y **TORSO** y los actuadores que se pueden utilizar con estos, se incluye la opción **ARRIBA**, que opera con la referencia **ROBOT** pero solo permite controlar

brazos y torso.

CSVMOCAP_Func.py incluye funciones para la extracción de las listas necesarias para la ejecución del movimiento del robot humanoide NAO, siendo estas la lista con actuadores involucrados, lista de tiempos de ejecución y la lista con los vectores de movimiento espacial, esta última incluye los vectores de movimiento de todos los actuadores en una sola lista organizada por actuador. En cuanto a la lista de tiempos de ejecución, se tiene almacenados en una lista los valores extraídos del MoCap, sin embargo, se crea a discreción otra lista con tiempos respecto a un coeficiente temporal, esto para permitir ejecutar la imitación ya sea más lento o más rápido de lo que se logra con los tiempos establecidos por la grabación. En general se pretenderá hacer el movimiento más lento que el realizado por la persona, ya que el NAO pierde el equilibrio muy fácil si realiza movimientos muy rápidos que no le permiten ajustar el balance a tiempo.

3.6. Movimiento del robot humanoide NAO imitando la grabación

La ejecución del movimiento del NAO como imitador de la grabación de MoCap se realiza por la ejecución de **Move.py**. Este realiza el llamado al ajuste de los datos con las funciones de **CSVMOCAP_Func.py** y les extrae para luego utilizar las listas con información como parámetros para las instrucciones del API ALMotion del NAO. Al inicio del programa se debe realizar la creación de proxies que permiten el uso de los módulos **ALMotion** y **ALPosture** para la manipulación del movimiento de cuerpo completo del NAO.

```
1 motionProxy = ALProxy("ALMotion", robotIP, PORT)
2 postureProxy = ALProxy("ALRobotPosture", robotIP, PORT)
```

También se realizan ajustes para la operación como la selección del marco de referencia a utilizar, según este se da la extracción de los datos con respecto a **TORSO** o **ROBOT**. Dependiendo del marco de referencia utilizado será la libertad de cadenas de actuadores a utilizar (revisar la sección 2.2.2 sobre el uso de cada marco de referencia). Se incluye también el parámetro de control de DoF, según el valor de este parámetro será la restricción de movimiento para las cadenas de actuadores, como una lista con la indicación de control por actuador. Si se ingresa la definición **AXIS_MASK_VEL** (propiedad de **ALMotion** con valor entero 7) se indica que se desea tener libertad de acción en 3 DoF, es decir X, Y y Z, si se utiliza más bien **AXIS_MASK_ALL** (con valor entero 63) se indica el uso de 6 DoF en el actuador de interés agregando la rotación wX, wY y wZ, y de modo similar con diferentes combinaciones de restricciones para los ejes. Estos valores están definidos por **AlMath** y **ALMotion**, y según el resultado de la suma del valor de la máscara de cada DoF se entenderá la restricción seleccionada.

```
1 ##AlMath
2 #define AXIS_MASK_X 1
3 #define AXIS_MASK_Y 2
4 #define AXIS_MASK_Z 4
5 #define AXIS_MASK_WX 8
6 #define AXIS_MASK_WY 16
```

```

7 #define AXIS_MASK_WZ 32
8
9 ##ALMotion
10 #define AXIS_MASK_VEL 7
11 #define AXIS_MASK_ALL 63
12
13 # *****
14 #Para usar solo X,Y,Z = 1+2+4 = 7 = AXIS_MASK_VEL
15 axisMask = almath.AXIS_MASK_X + almath.AXIS_MASK_Y + almath.AXIS_MASK_Z

```

En el caso en que no se utilizan los valores de rotación, las instrucciones del NAO de igual manera operan esperando datos para 6 DoF por posición para cada actuador, de modo que si no se dispone de datos de rotación se debe pasar algún valor para wX, wY y wZ aunque no sea utilizado, se ha decidido enviar el valor numérico "0.0." en las rotaciones mientras no se estén utilizando. Se debe asignar una restricción por actuador, no siendo estos necesariamente iguales, aunque el sistema por defecto utilizará la misma restricción para cada actuador.

```

1 axisMask = [ motion.AXIS_MASK_VEL ]*6

```

Los parámetros usados para la instrucción de movimiento deben corresponder por orden de elemento en las listas, por ello la importancia de acomodar los datos por actuador en pasos previos. Es decir, el primer elemento de la lista de tiempos debe ser el correspondiente al primer elemento de la lista de vectores (X,Y,Z,wX,wY,wZ), que a su vez deben ser los respectivos para la primera cadena de actuadores enlistada (en el caso preferente sería **RArm**). Para este actuador debe corresponder su respectiva máscara de restricción de DoF. En el caso del marco de referencia se utiliza un solo valor que se define para todos los actuadores en uso, con valor definido por **ALMotion**. Este formato se debe cumplir para la instrucción *positionInterpolations()*, si se utiliza alguna otra se deberá hacer el cambio adecuado en el proceso de acomodo de datos para que estos calcen con lo que se solicite, aunque suelen ser iguales.

El último parámetro que recibe la instrucción corresponde al uso de valores de posicionamiento absolutos o relativos.

```

1 #Se envia la grabacion entera
2 motionProxy.positionInterpolations(listaActuadores, referencia, listaCoordenadas, ←
    axisMask, listaTiempos, absolutos)

```

Es importante tener en cuenta que *positionInterpolations()* es una instrucción bloqueante, de modo que si se pasa como parámetros los vectores de la grabación completa, el único control de balance hasta que se concluya su ejecución será la de balance automático del NAO. La alternativa de ajustar las restricciones del balanceador automático según el posicionamiento del COM del robot NAO se debe hacer con una pequeña modificación, esto es recorrer la lista de vectores cada cierta cantidad de cuadros y realizar el ajuste según la ubicación del COM antes de pasar al siguiente grupo de cuadros. Como se intenta mover todas las cadenas al mismo tiempo se supone que las listas de coordenadas para cada actuador tienen la misma longitud, de lo contrario sucedería que algunos actuadores llegarían al final de

su ejecución antes que otros, lo que puede causar problemas de balance, correlación del movimiento con la persona o de ejecución del código. Aquí radica la importancia de revisar los archivos CSV a ejecutar y eliminar las filas en que se pierde algún marcador/cuerpo rígido (según el modelo) y mantener todas las longitudes de datos iguales entre actuadores.

```

1 #Caso enviando por grupos de cuadros (fps)
2 for i in range(0,len(listaCoordenadas[0]),fps):
3     motionProxy.positionInterpolations(listaActuadores, referencia,[←
        listaCoordenadas[0][i:i+fps-1],listaCoordenadas[1][i:i+fps-1],←
        listaCoordenadas[2][i:i+fps-1],listaCoordenadas[3][i:i+fps-1]],←
        axisMask,[listaTiempos[0][0:len(listaCoordenadas[0][i:i+fps-1])],←
        listaTiempos[1][0:len(listaCoordenadas[1][i:i+fps-1])],←
        listaTiempos[2][0:len(listaCoordenadas[2][i:i+fps-1])],←
        listaTiempos[3][0:len(listaCoordenadas[3][i:i+fps-1])]],absolutos)
```

Donde *listaCoordenadas[0][i:i+fps-1]* corresponde a las coordenadas de ubicación espacial para un grupo de cuadros de grabación para un solo actuador. *listaCoordenadas* incluye toda la información de posicionamiento espacial de todos los actuadores, según se haya especificado con **ROBOT**, **TORSO** o **ARRIBA**.

Para enviar los datos de ubicación en grupos de *N* cuadros se utiliza la notación de cuadros por segundo (*FPS*), se usa la instrucción bloqueante y se envía un rango de FPS cantidad de cuadros por cada actuador.

```

1 [listaCoordenadas[0][i:i+fps-1],listaCoordenadas[1][i:i+fps-1],listaCoordenadas←
  [2][i:i+fps-1],listaCoordenadas[3][i:i+fps-1]]
```

Estos rangos deben corresponder con el rango de las listas de tiempos que se reciben por cada actuador.

```

1 [listaTiempos[0][0:len(listaCoordenadas[0][i:i+fps-1])],listaTiempos[1][0:len(←
  listaCoordenadas[1][i:i+fps-1])],listaTiempos[2][0:len(listaCoordenadas[2][i:i←
  +fps-1])],listaTiempos[3][0:len(listaCoordenadas[3][i:i+fps-1])]]
```

Al enviar los datos de esta manera se debe considerar que cada paquete de instrucciones de movimiento es independiente, de modo que el contenido de las listas de tiempos para cada grupo que se envía debe comenzar siempre en el primer segundo de la animación (>0), si se envían los grupos correspondientes según como estaban acomodados para enviar toda la animación en un solo llamado a la instrucción, existirán bloques sin acción mientras el temporizador del nuevo llamado llega al valor inicial de la lista de tiempos. Este caso es más fácil de comprender con un ejemplo.

Supóngase que la animación a imitar por el NAO tiene una duración total de 10 s, exportado a 30 FPS desde Motive (es importante mantener los FPS bajos ya que el MoCap puede trabajar a mayor resolución de FPS que la capacidad de ejecución del NAO, generando más cantidad de cuadros de posicionamiento que no aportan al movimiento observado en el NAO), dando un total de 300 conjuntos de datos que incluyen posición (X,Y,Z,wX,wY,wZ) para cada actuador, con su respectivo momento de

ejecución en segundos indicado en los elementos de la lista de tiempos. Al haber 30 FPS se entiende que habrá un cuadro de posicionamiento cada $\approx 0,0333$ s, por lo que la lista de tiempos completa para cada actuador tendrá la forma [0.0333, 0.0666, ..., 10]. Si se decide hacer el llamado a la tarea de ejecución del movimiento para grupos de 30 datos (30 FPS) recorriendo la lista completa, los primeros dos bloques en enviarse serían [0.0333, 0.0666, 0.1332, ..., 1] y [1.0333, 1.0666, 1.1332, ..., 2], siguiendo con la línea temporal de la grabación, de modo que el primer bloque se ejecutaría como se esperaría, pero el segundo bloque, al ser independiente del primero, pondrá el temporizador de la ejecución del movimiento del NAO de vuelta en 0 s, y ejecutará su acción hasta llegar al segundo 1.0333. Es decir, luego de que el primer bloque se ejecute habrá 1.0333 s de espera hasta que suceda el primer cambio de posición del segundo bloque. Este efecto se nota más en la ejecución del movimiento mientras más bloques hayan pasado, cuando se esté enviando el último que comienza en 9.0333 s, habrá una pausa de 9.0333 s antes de que este se ejecute.

Esto genera un movimiento cortado de la animación, con mayor duración. Para evitarlo se debe re establecer el inicio de la lista, para el ejemplo, enviar [0.0333, 0.0666, 0.1332, ..., 1] para cada actuador en cada grupo de cuadros que se envía para ejecutar.

Este modelo de enviar grupos de cuadros de posicionamiento sirve como base para una próxima implementación de teleoperación en *tiempo real*, ya que se reciben paquetes de datos con la información de cada cuadro en grabación desde Motive. Es posible entonces almacenar varios cuadros recibidos, procesarlos el ajuste, y luego enviarlos en un llamado individual de ejecución del movimiento, y repetir el proceso mientras se reciba la información de Motive. Siempre considerando el caso descrito anteriormente para las listas de tiempos de ejecución.

Luego de haber realizado los ajustes necesarios a los datos para realizar el movimiento, se inician los preparativos de posicionamiento del robot NAO para ejecutar la imitación de la grabación. Estos pasos son:

1. Activación de la rigidez de los motores (a cuerpo completo indicado con **Body**)
2. Posicionamiento del robot humanoide NAO a pose **Stand**
3. Inabilitación del administrador de caídas
4. Habilitación del control automático de balance de cuerpo completo
5. Definición de las restricciones de balance
6. Activación del balance de cuerpo completo con las restricciones definidas
7. Inicio y finalización del movimiento del robot humanoide NAO
8. Habilitación del administrador de caídas
9. Inabilitación del balance de cuerpo automático
10. Posicionamiento del robot humanoide NAO a pose **Crouch**
11. Poner motores en reposo

Para ejecutar **Move.py** se asume que todos los archivos CSV necesarios ya han sido organizados según los formatos descritos anteriormente en este capítulo, y que ya se realizó la calibración. Además, se debe saber de antemano la dirección IP del robot humanoide NAO que se desea teleoperar (pulsar el botón en el pecho del NAO una vez) considerando que la PC y el NAO se encuentran conectados en la misma red local. Por defecto se utiliza el puerto 9559. Se debe saber también el nombre del archivo CSV con los datos de la grabación que se quiere ejecutar, y tener este archivo localizado en el directorio del proyecto .../*Choreography*/, así como los offsets en .../*Calibration/Offsets*/ (de esto se encarga el código). Importante recordar que los datos de calibración de la persona deben estar dentro de .../*Calibration/Human*/, ya sea directo o dentro de alguna sub-carpeta (el proyecto parte con la carpeta *RigidBody* con los datos de una persona que fueron usados para las pruebas de funcionamiento del sistema), con los nombres correspondientes a la rutina (ver Apéndice B), y los del NAO en .../*Calibration/NAO/RigidBody_Default* (el proyecto parte con esta base), o bien en alguna carpeta dentro de .../*Calibration/NAO*/, que se debe especificar para la lectura al correr la calibración.

3.7. Uso del sistema de teleoperación paso a paso

A continuación una lista de pasos con la ejecución del sistema de teleoperación, se entiende que para ejecutar estos pasos sin problemas se han leído las secciones del capítulo 3 anteriores a la presente sección y que se entienden los conceptos explicados en el capítulo 2.

3.7.1. Preparación del ambiente de trabajo

A continuación el procedimiento para preparar el ambiente de trabajo que se debe tener previo al uso del sistema de teleoperación.

El proyecto ha sido desarrollado con Linux como sistema operativo, de modo que para la preparación de la estación de trabajo se hace la suposición que se está en ese medio, esto para la instalación de las diferentes herramientas que se ocupan en el sistema. Solo el caso de Motive se supone que se utiliza con sistema operativo Windows.

Además, se supone que al clonar el git del proyecto se está usando la versión liberada **Ver_Release**.

Choregraphe

Choregraphe es utilizado para correr las rutinas de calibración del NAO. Estas rutinas se encuentran en el directorio del proyecto .../*Code/Ver_Release/Choregraphe_Base*/ . Cada rutina está separada en una carpeta independiente con el nombre de la rutina, estos son **Brazos**, **PiernaD**, **PiernaI**, **Torso** y **Cabeza**, con la extensión **pml**.

Para instalar Choregraphe [5], si se tiene el CD de documentación y software del NAO (viene incluido con el robot) solo basta extraer el archivo **choregraphe-suite-x.x-linux32.tar.gz**, dentro se encuentra el instalador. También se puede descargar de la página de ALDEBARAN <https://community.ald.softbankrobotics.com/>, para realizar la descarga se debe tener una cuenta registrada. Luego de instalado se puede utilizar el programa con una prueba gratis de 90 días, o bien ingresar la licencia (si se usó CD el código viene impreso en la caja del disco).

Una vez disponible el programa la primer prueba de funcionamiento es conectarse al robot humanoide NAO. En la barra de acciones se presiona el ícono de conexión (figura 3.3) y se verifica que dentro de las opciones aparezca reconocido el robot NAO con dirección IP y nombre. Aparecerá también otro robot que es utilizado para simulaciones, el cual es virtual y local a la computadora, si no se detecta al robot NAO real que se va a utilizar, solo se mostrará el virtual local.

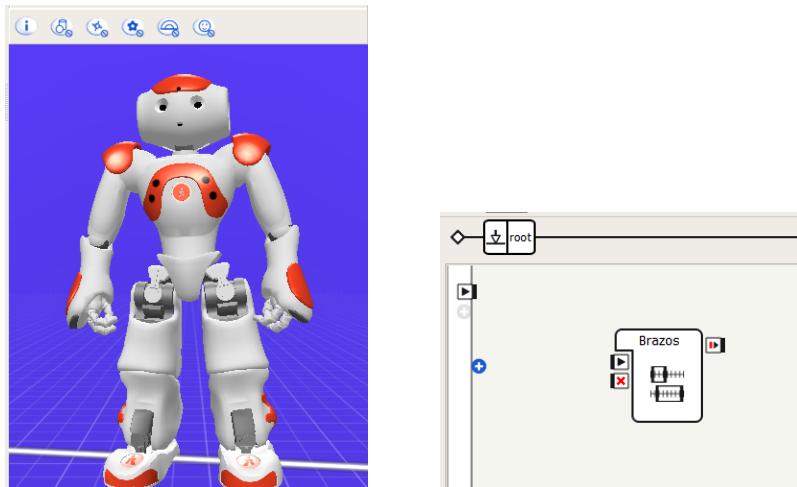


Figura 3.3: Barra de acciones de Choregraphe con ícono de *realizar conexión*

Para probar que el NAO está conectado a la red se puede presionar el botón en centro de su pecho y esperar a que el NAO diga su dirección IP. Se puede verificar el acceso remoto al robot escribiendo la dirección IP en un buscador web, esto cargará la página web local del robot NAO, donde se puede configurar al mismo. Este paso es importante ya que se requiere para habilitar la desactivación del administrador de caídas (sección 3.7.4).

Al conectarse al NAO por medio de Choregraphe se desplegará un modelo 3D del robot NAO con la posición actual del robot (ver figura 3.4a).

Una vez conectado se deberá cargar la rutina a ejecutar especificando el directorio donde se encuentra y abriendo el archivo con extensión *.pml*. Cuando cargue la rutina se verá un bloque con la misma en Choregraphe, como en la Figura 3.4.



(a) Visualizador virtual del NAO conectado

(b) Bloque de rutina de calibración *Brazos* cargado en Choregraphe

Figura 3.4: Interfaz Choregraphe con NAO conectado y rutina cargada

Python

Primero se debe tener el ambiente de Python con la versión 2.7.14 (versión usada para el desarrollo del proyecto). Se debe incluir el paquete matemático *numpy* para poder efectuar la regresión lineal con la función *polyfit*.

```
$ sudo apt-get install python-2.7 python-numpy
```

Además del ambiente de programación en Python se debe incluir el SDK oficial de ALDEBARAN para utilizar el robot humanoide NAO con Python. Este Python SDK [13] se obtiene de la página de descarga <https://community.ald.softbankrobotics.com/>, para la que se ocupa accesar con usuario registrado. También es posible extraer los archivos del SDK desde el CD (mismo proceso para C++ SDK).

Luego de tener el SDK extraído se debe agregar a la variable de entorno **PYTHONPATH** el directorio donde se puso el SDK, para que pueda ser encontrado cada vez que se trata de correr código que utiliza los API's del NAO cuando se importa NAOqi² a un script.

Para corroborar que el ambiente de desarrollo para el NAO con Python está preparado se puede realizar la siguiente prueba en Python por terminal.

```
$ python
>>> import sys
>>> import naoqi
```

No debería ocurrir algún error al ejecutar los comandos anteriores. Para solucionar posibles errores se puede revisar la guía en línea de solución de errores de ALDEBARAN <http://doc.aldebaran.com/1-14/dev/python/tips-and-tricks.html#python-sdk-troubleshooting>.

Motive - MoCap

Para utilizar el MoCap se debe descargar e instalar Motive de la página oficial de OptiTrack <http://optitrack.com/products/motive/>. Es importante recordar que Motive corre en Windows.

Para poder iniciar una sesión de captura de movimiento se debe asegurar que las cámaras que conforman el sistema están instaladas en el espacio de grabación (Figura 2.4) y conectadas a la PC en que se usará Motive. Las configuraciones de los marcadores en el traje especial son según el modelo seleccionado para la grabación (sección 2.3). El proceso de calibración y preparación del MoCap se puede revisar en el Manual de Uso del PRIS-Lab [45], o bien de las instrucciones detalladas en el wiki de Optitrack https://v20.wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki.

Además de disponer del equipo para la grabación y el software Motive, se debe tener las Licencias físicas para poder utilizar Motive, estas deben ser compradas a OptiTrack.

²<http://doc.aldebaran.com/1-14/dev/naoqi/index.html>

Repository del proyecto

El proyecto cuenta con un repositorio git https://github.com/LennonNM/Proyecto_Electrico, donde se incluye la última versión del código del sistema de teleoperación, con los archivos CSV necesarios para su primer uso, así como de diferentes archivos con resultados comparativos y ejemplos de formatos para comparar los resultados de los datos obtenidos del NAO y del MoCap por medio de gráficas (archivos de hojas de cálculo) donde solo se debe ingresar los datos del NAO en la hoja *Datos_NAO* y los del MoCap en *Datos_MoCap*; también se incluye una carpeta de desarrollo *Dev_Files* que incluye las versiones anteriores generadas por el proceso de desarrollo, las versiones de desarrollo se identifican por las carpetas **Ver#**, donde # corresponde al número de versión, estas versiones son solo de referencia para consultar ideas previas al sistema final; algunos archivos CSV de grabaciones del MoCap utilizadas para las pruebas iniciales y varios ejemplos en Python extraídos de la página de ALDEBARAN para el control de cuerpo completo del NAO.

Todo lo anterior se encuentra dentro de la carpeta *.../Code*. En la carpeta *.../Info* se incluyen links a fuentes consultadas y otros misceláneos como un *README*. Todo el código está documentado.

Para correr el código se debe ingresar al directorio con la versión más reciente *.../Code/Ver_Release/* y correr ya sea la calibración o la teleoperación, con los parámetros requeridos según sea el caso de la configuración que se desea.

```
$ python Calibrate.py 2 yes humanDir
$ python Move.py 10.0.1.128 grabacion.csv ROBOT
```

En terminal se desplegarán indicaciones del proceso de los diferentes pasos que ejecuta cada código, esto sirve para encontrar errores de implementación o uso, así como para saber por dónde va ejecutado el código.

3.7.2. Generación de los datos de calibración del NAO

Para generar los archivos CSV con los datos del movimiento del NAO para la calibración se debe tener primero las rutinas creadas (ubicadas en *.../Code/Ver_Release/Calibration/Routines*), para luego ejecutarlas por medio Choregraphe en el NAO. Es importante que el robot NAO se encuentre conectado a Choregraphe y esté activo para moverse (presionar **wake up** de la barra de acciones, figura 3.5), de lo contrario no se moverá aunque la rutina se cargue al robot. Si se está creando una rutina en *Timeline* de Choregraphe, se debe activar el modo animación.



Figura 3.5: Barra de acciones con ícono *wake up* (extremo derecho) y activación del modo animación (extremo izquierdo)

Se debe tener una terminal abierta con la cual ejecutar **GetPositions.py**, esta se debe ejecutar antes de enviar la rutina al NAO por Choregraph. Tanto la rutina de calibración como **GetPositions.py**

deben estar en ejecución al mismo tiempo. Para detener la captura del NAO se presiona *Ctrl+C* en terminal.

```
$ python GetPositions.py 10.0.1.18 ROBOT yes
```

No debería ser necesario realizar la captura de los datos del NAO para las rutinas preestablecidas, ya que los archivos CSV del NAO ya existen. Sin embargo, se pueden volver a realizar, estos archivos se generarán en *.../Code/Ver_Release/Calibration/NAO/GetPositions_Generated/*, se les cambia el nombre para que corresponda con los esperados para la calibración **Brazos.csv**, **PiernaD.csv**, **PiernaI.csv**, **Torso.csv** y reubicarlos a *.../Code/Ver_Release/Calibration/NAO/Carpeta_Deseada*, especificando esta nueva carpeta al realizar la calibración, con cuidado de no sobreescribir los que existen en *.../Code/Ver_Release/Calibration/NAO/RigidBody_Default* que corresponden a los archivos por defecto del sistema base.

Una vez con los datos del movimiento del NAO se procede a capturar los de la persona en MoCap para la calibración, se debe asegurar que el NAO esté con sus emisores LEDs cubiertos o que esté ubicado en algún lugar donde no interfiera con el rastreo de los marcadores. Por medio de Choregraphe se ejecutan nuevamente las rutinas (no es necesario tomar los datos del NAO con **GetPositions.py** en este paso, se recomienda que se tomen primero los del NAO y luego los de la persona en momentos distintos y no al mismo tiempo, esto por si hay algún fallo en la grabación del MoCap que no afecte la toma de los del NAO). La persona debe imitar los movimientos ejecutados por el robot humanoide NAO tratando de cumplir con los mismos tiempos de ejecución. La grabación debe iniciar cuando el NAO se comience a mover (la persona debe comenzar a moverse igual) y terminar cuando el NAO regrese a la posición inicial **Stand**.

Al exportar las grabaciones de Motive se debe tomar en cuenta usar 30 FPS, y las diferentes opciones de información a exportar necesaria según el modelo usado. Si se usa el modelo de 6 marcadores se deberá habilitar los datos de marcadores solamente, si se usa el modelo con cuerpos rígidos se habilita los datos del cuerpo rígido solamente, y si es el modelo de esqueleto humano se deberá habilitar solo los datos de esqueleto (ver figura *refF:exportMotive*). Además, para obtener un buen acople de curvas se requiere revisar los datos exportados de Motive contra los datos capturados del NAO, para esto se pueden usar los archivos de hojas de cálculo ubicadas en *.../Code/Ver_Release/Comparisons/Template*, organizados por rutina de calibración, en las hojas *Datos_NAO* y *Datos_MoCap* del archivo se debe sustituir los valores según las columnas correspondientes y las gráficas se actualizarán con estos valores. Las gráficas sirven para observar cómo calzan los puntos entre sí y poder seleccionar cuáles cuadros de la grabación son los que corresponden con los datos del NAO para la regresión, por medio del rango de datos, para modificar los CSV según se requiera. Se puede usar cualquier otro método de graficación para visualizar los datos de los CSV.

Se debe eliminar los datos de los elementos de los archivos de la persona que no sirven para la calibración (escogidos por proceso visual manual con ayuda de la graficación de los datos para la selección de los rangos de datos a incluir), en su lugar se debe ingresar algún carácter o palabra, ya que esta es detectada en el módulo de calibración e ignorada, manteniendo el orden en los datos completos del archivo y permitiendo filtrar estos datos que no se usarán. De referencia se incluye la palabra *Empty* en

estas casillas (ver Figura 3.6). Si los datos calzan correctamente en sus rangos completos se omite este paso.

Coordinate Space	Global										
Rigid Body	Rigid Body	Rigid Body	Rigid Body	Rigid Body	Rigid Body	Rigid Body	Rigid Body	Rigid Body	Rigid Body	Rigid Body	Rigid Body
RArm	RArm	RArm	RArm	RArm	RArm	RArm	RArm	RArm	LLeg	LLeg	LLeg
39740428CB1B11	39740428C	39740428C	460E2DC9	460E2DC9	460E2DC9						
Rotation	Rotation	Rotation	Rotation	Position	Position	Position	Error Per M	Rotation	Rotation	Rotation	Rotation
X	Y	Z	W	X	Y	Z		X	Y		
Empty	Empty	Empty	-0.808581	Empty	Empty	Empty	0.00064	0.002014	-0.089615		
Empty	Empty	Empty	-0.814482	Empty	Empty	Empty	0.000647	0.002119	-0.089408		
Empty	Empty	Empty	-0.820435	Empty	Empty	Empty	0.000679	0.002063	-0.089649		
Empty	Empty	Empty	-0.82556	Empty	Empty	Empty	0.000708	0.001968	-0.089662		
Empty	Empty	Empty	-0.826295	Empty	Empty	Empty	0.000669	0.001926	-0.089681		
Empty	Empty	Empty	-0.827632	Empty	Empty	Empty	0.000753	0.00186	-0.089765		
Empty	Empty	Empty	-0.829652	Empty	Empty	Empty	0.000821	0.001829	-0.089774		
Empty	Empty	Empty	-0.831394	Empty	Empty	Empty	0.000872	0.001922	-0.089704		
Empty	Empty	Empty	-0.832396	Empty	Empty	Empty	0.000949	0.001936	-0.089706		
Empty	Empty	Empty	-0.83411	Empty	Empty	Empty	0.000952	0.001952	-0.089718		
Empty	Empty	Empty	-0.836266	Empty	Empty	Empty	0.000972	0.00177	-0.089808		

Figura 3.6: Elementos del archivo CSV de una persona correspondientes al actuador NAO eliminados y reemplazados por *Empty* como parte del proceso de calibración

3.7.3. Calibrar el sistema de teleoperación

Para obtener los coeficientes del ajuste polinomial se debe ejecutar **Calibrate.py**, considerando que ya se acomodó los datos del MoCap para que calcen en el tiempo con los del NAO.

```
$ python Calibrate.py 2 yes humanDir
```

Se puede recurrir a las hojas de cálculo con las gráficas de los datos para observar la curva corregida, incluyendo los coeficientes del ajuste en las casillas bajo las correcciones a cada coordenada (X Corr, Y Corr, Z Corr, wX Corr, wY Corr, wZ Corr), según el grado del polinomio utilizado. Luego de generados los coeficientes debería verse los resultados almacenados en el archivo .../Code-/Ver_Release/Calibration/Offsets/offsets.csv en el caso por defecto, o bien el seleccionado por el usuario.

3.7.4. Teleoperar el robot humanoide NAO

Para teleoperar el robot humanoide NAO tienen que existir todos los archivos CSV que ocupa el sistema (sección 3.2) y estar ubicados donde corresponde. Además, se debe haber habilitado la desactivación del administrador de caídas, de lo contrario este se podrá activar en medio de la ejecución de la imitación de la grabación.

Para habilitar la desactivación del administrador de caídas se debe ingresar a la página de configuraciones del robot a teleoperar, ingresando la dirección IP del robot en un buscador web. Desde la página web del NAO se debe hacer click en el símbolo de pregunta "?" en la barra de herramientas, al lado derecho. De las opciones que se despliegan se debe seleccionar **Former robot webpage**, esta opción permite ingresar a configuraciones avanzadas del NAO, los pasos se detallan en la página de

documentación de acceso a las opciones avanzadas: <http://doc.aldebaran.com/2-1/nao/nao-webpage.html#opennao-web-page>. Se ocupa contraseña para ingresar.

Una vez dentro de las opciones avanzadas se debe seleccionar la opción **Settings** en la barra de herramientas y marcar la casilla bajo la opción **Fall manager reflex**, esto habilita la desactivación del administrador de caídas. Se desplegará una ventana de confirmación de la acción, hacer click sobre **I understand the risks** y estará lista la habilitación. La desactivación/activación se realiza en el código de **Move.py**.

```
1 motionProxy.setFallManagerEnabled(False)
```

La teleoperación inicia al ejecutar **Move.py** con los parámetros necesarios.

```
$ python Move.py 10.0.1.128 grabacion.csv ROBOT
```

Durante esta ejecución se debe tener precaución de los movimientos que ejecute el NAO para atenderle ante caídas. El movimiento iniciará con el NAO en la pose predeterminada **Stand** y terminará con una pausa de su movimiento seguido del posicionamiento en **Crouch**. Una vez en esta pose ha terminado la teleoperación con imitación de la grabación del MoCap. En terminal se desplegarán mensajes sobre el proceso de la teleoperación.

Capítulo 4

Validación del sistema de teleoperación

La validación del sistema se conforma de dos revisiones en el movimiento final ejecutado por el NAO. Primero se debe corroborar la intención del movimiento, por ejemplo, si el operador humano levantó el brazo derecho en cierto momento de la rutina se espera que el robot humanoide NAO realice la misma acción (sección 4.1). La segunda revisión consiste en la evaluación de los datos utilizados por el sistema de teleoperación (sección 4.2), estos datos corresponden a los vectores de ubicación espacial de cada actuador final de las cadenas de acción involucradas.

Los resultados completos (imágenes y gráficas) se encuentran en el Apéndice A.

La validación se enfoca en el movimiento de los brazos del robot humanoide NAO en respuesta a la coreografía realizada. Sin embargo, se realizó pruebas para el movimiento complejo del robot NAO (movimiento de Tórso y Piernas) para dejar lista una base de pruebas para trabajos a futuro cuyo enfoque sea el balance del NAO para estos movimientos complejos, estos resultados se muestran en el Apéndice A.

Se utilizó parte de las mismas rutinas de calibración para las pruebas iniciales de comparación del movimiento ejecutado, además se realizó las rutinas *Rutina_Círculos_1* y *Círculos-con-Lateral*, cuyos datos exportados en CSV se encuentran en la carpeta *.../Code/Ver_Release/Choreography*. Las grabaciones en MoCap se realizaron para dos individuos diferentes, identificados con **Sujeto1** y **Sujeto2**, los proyectos de Motive completos de estas rutinas se encuentran en el directorio *.../Dev_Files/Test_Data_MoCap*.

Para los resultados gráficos se utilizó las rutinas de calibración, ya que se tenía disponible los datos de los sensores de los actuadores del NAO. Se utilizó tanto el ajuste por regresión lineal con polinomio de grado 1 como el de grado 2, se muestran los resultados solo para el polinomio de grado 1; en general para ambos casos el peor ajuste se dio en las rotaciones del actuador, esto debido a la mala relación existente entre la capacidad del robot humanoide NAO de controlar la rotación de sus actuadores con la del ser humano, que involuntariamente puede provocar movimientos que *ensucien* la toma de datos. Además, se dan casos en que se nota *vibración* en los marcadores capturados por el MoCap, esto debido a movimientos naturales del cuerpo (respiración, balanceo, movimientos involuntarios al estar mucho tiempo de pie, entre otros posibles), que el traje no estaba lo suficientemente ajustado como para que los marcadores no se corrieran mientras se efectúa un movimiento, el mismo cálculo de la posición de los marcadores por Motive, entre muchas otras posibles causas, es decir, el MoCap es capaz de capturar toda esta información y reflejarla en los datos de movimiento, pero el NAO no es capaz en primera

instancia de generar esto en sí mismo, y la única información capturada es una aproximación de la posición de sus motores por medio de los sensores.

Por ello se ve una curva más *limpia* en el caso de las capturas del NAO, y se ve más picos en las capturas del MoCap. Estos pequeños picos y distorsiones son tomados en cuenta cuando se realiza la regresión lineal, ya que no ha sido implementado un filtro o atenuador de curvas, de modo que si la información del MoCap llega *sucia*, la calibración así mismo se *ensuciará*.

4.1. Confirmación de la intención del movimiento

A continuación se muestran fotografías de la pose realizada por una persona en MoCap y la ejecución de imitación realizada por el NAO al ser teleoperado por el sistema actual.

4.2. Evaluación de los datos usados por el sistema de teleoperación

A continuación se muestran gráficas comparando los datos obtenidos del MoCap de una persona, los datos obtenidos de los sensores del NAO y el ajuste obtenido por el módulo de calibración. Las rutinas utilizadas para generar estos datos son las de calibración, los datos del NAO son de los respectivos CSV por defecto usados en el proceso de calibración normal del sistema. Se muestran los resultados solo para la cadena RArm, los resultados completos se muestran en el apéndice A.

Se puede observar que en el caso de la Figura 4.2, a pesar de que se acomodaron de manera correcta las curvas en el tiempo, existe en los datos del MoCap un segundo valle bastante pronunciado, esta información toma parte en los cálculos del ajuste *ensuciando* el resultado y generando un mal acople para este eje.

En la Figura ?? se puede observar más bien la no relación entre los datos debido a posibles giros involuntarios que se realizaron alrededor de ese eje, giros que el NAO controla con facilidad pero para los que la persona puede fallar.

Se ve también otros casos como en la Figura ??, en que hay relación de movimiento pero la curva está invertida en la intención del movimiento, esto se repite en otros ejes (como el X) debido a las diferencias entre las direcciones de los ejes de los marcos de referencia del robot humanoide NAO con respecto al del MoCap, sin embargo, dado que hay una correcta relación del movimiento ejecutado y están correctamente acomodados, esta inversión se puede considerar como solo una multiplicación de los datos por -1, de modo que la regresión es capaz de capturarla y aún así realizar un ajuste comparable.

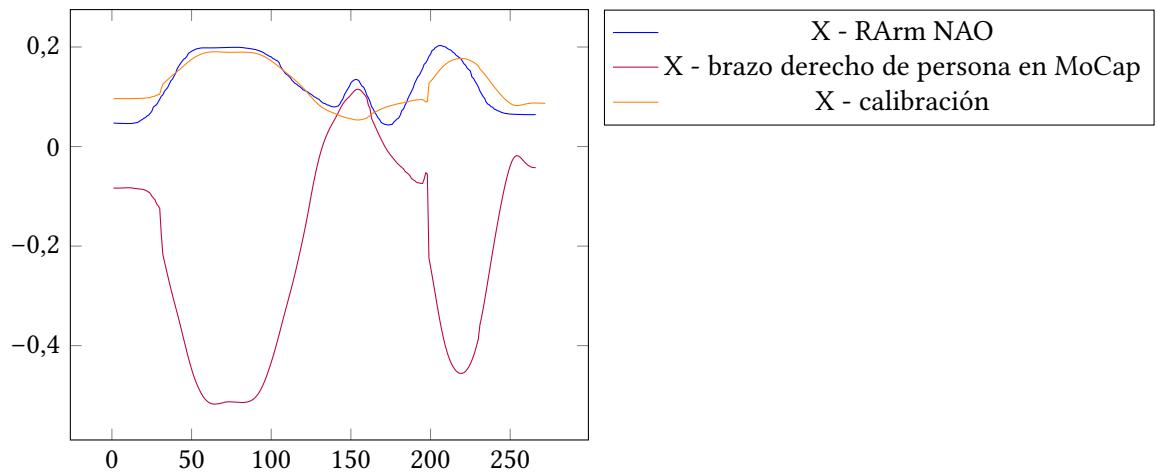


Figura 4.1: Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

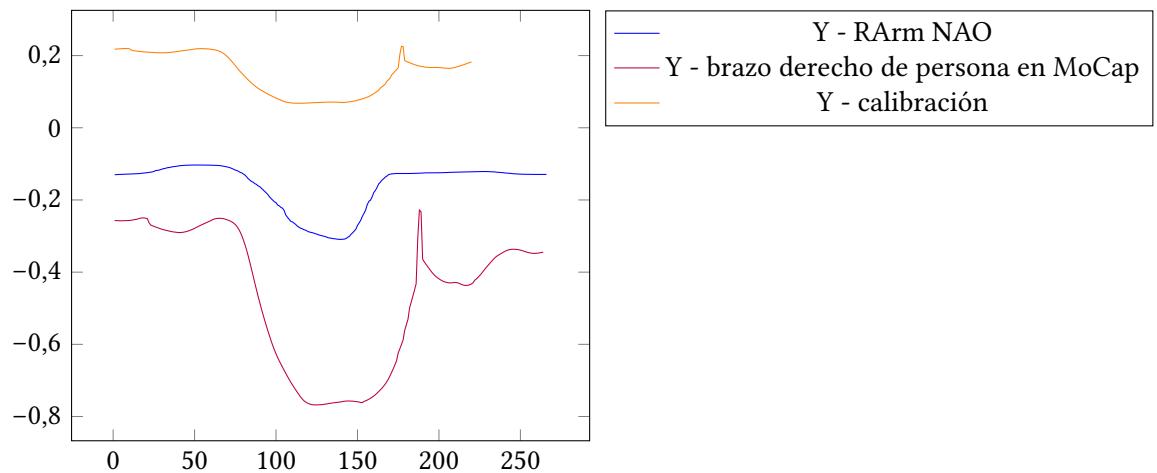


Figura 4.2: Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

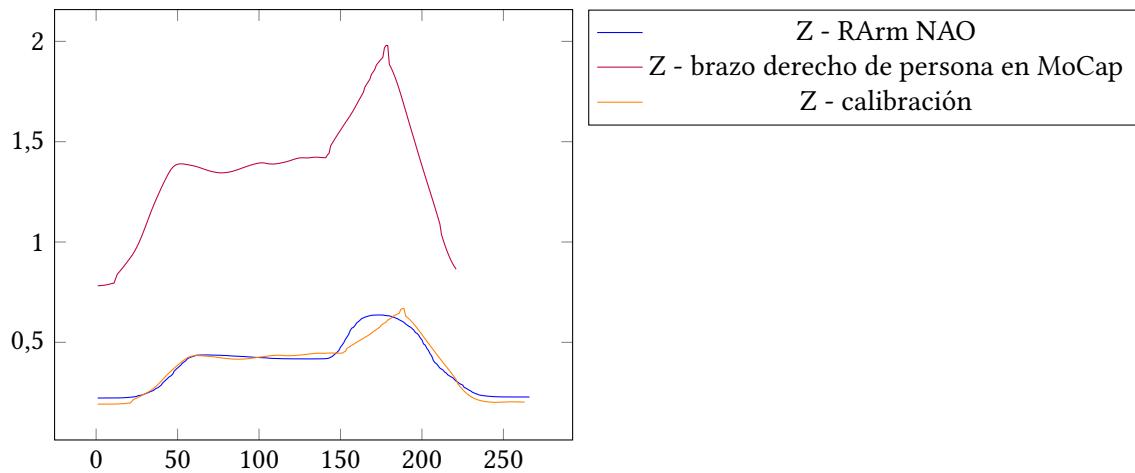


Figura 4.3: Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

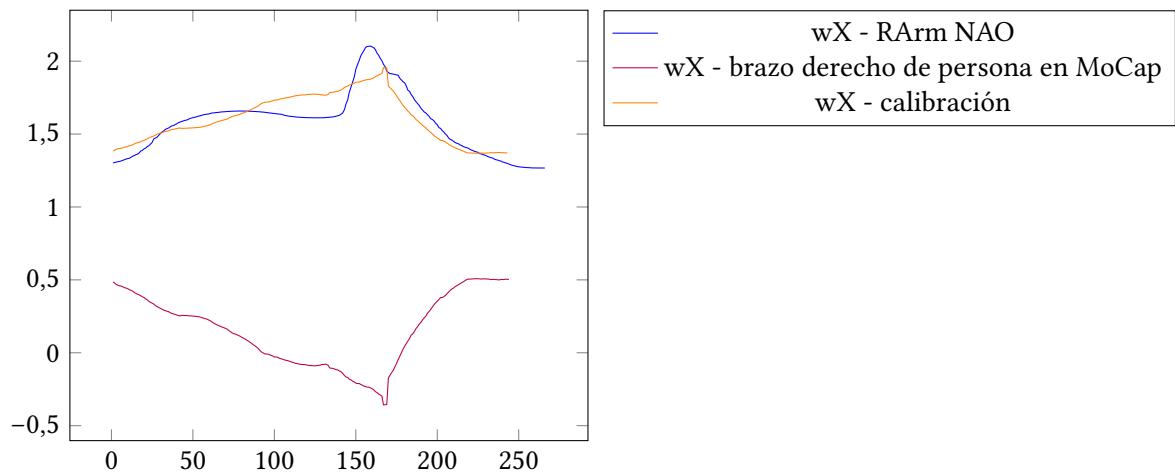


Figura 4.4: Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

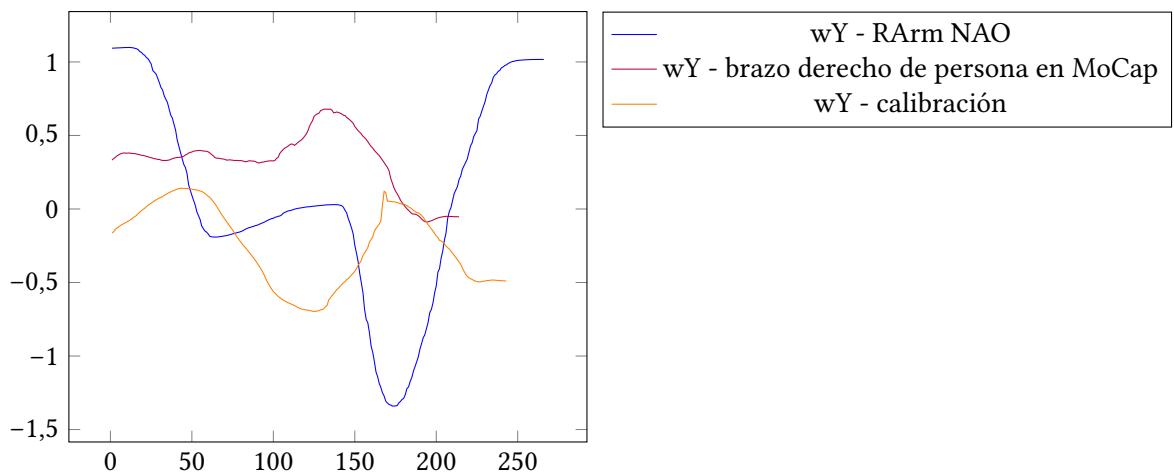


Figura 4.5: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

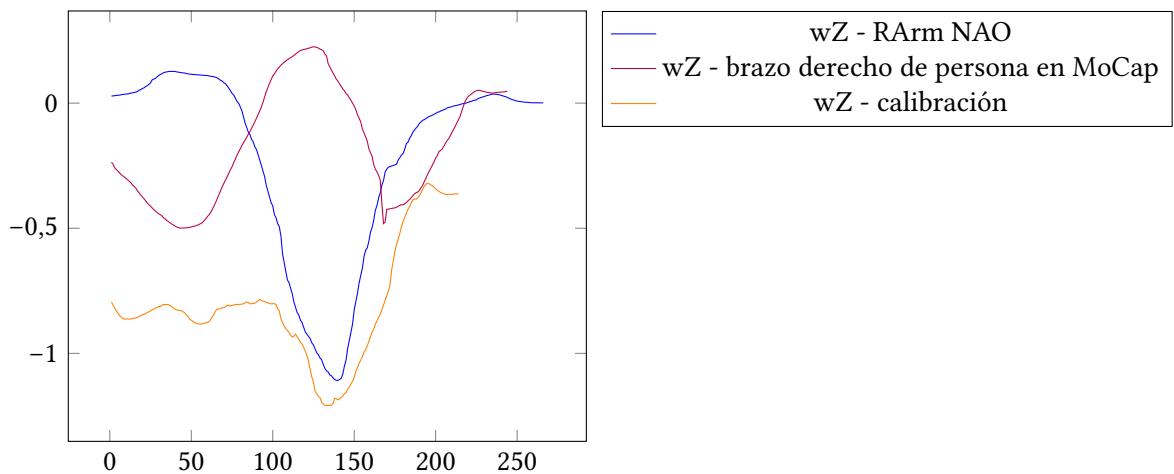


Figura 4.6: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

Capítulo 5

Conclusiones y recomendaciones

5.1. Conclusiones

- Se logró comprender la generalidad de la teleoperación, diseñando un modelo de las partes que conforman un sistema de teleoperación básico.
- Se diseñó un sistema teleoperación para un robot humanoide NAO utilizando un sistema de captura óptica de movimiento como fuente de datos de control.
- Se logró realizar capturas ópticas de movimiento de diferentes personas por medio del MoCap con cámaras Prime 41 y el programa Motive.
- Se adquirió mayor conocimiento sobre el control del robot humanoide NAO enfocado en el movimiento de cuerpo completo en los diferentes modos de acción disponibles.
- Se implementó la primer versión del PRIS-Lab de un sistema para la teleoperación de un robot humanoide NAO a través de grabaciones de captura óptica del movimiento de una persona.
- Se diseñó e implementó módulos para facilitar el uso del sistema de teleoperación en las secciones de calibración, extracción de datos de los sensores del NAO y de control del movimiento del robot humanoide NAO.
- Se logró teleoperar el robot humanoide NAO a partir de los datos de una grabación en MoCap de diferentes personas de manera satisfactoria para el movimiento de los brazos manteniendo el balance en el NAO.
- Se logró comprobar una correcta relación entre la intención del movimiento de los brazos del robot humanoide NAO generado por el sistema de teleoperación con respecto al movimiento de los brazos de la persona grabado en MoCap con evidencia visual.
- Se logró comprobar de manera analítica los resultados del ajuste de coordenadas de posicionamiento espacial de los actuadores del robot humanoide NAO por medio del proceso de calibración.

- Se implementó la base para proyectos de continuación y mejora del sistema de teleoperación presentado enfocado en realizar movimientos complejos de balance para el robot humanoide NAO.
- Se generó un documento escrito con toda la información recopilada y una guía para poder desarrollar un sistema de teleoperación básico desde cero.
- Se recopiló evidencia de las pruebas realizadas en proyectos de Motive, archivos CSV con los datos exportados de las grabaciones en MoCap, comparaciones gráficas en archivos de hojas de cálculo y video grabaciones de los procesos de captura de los datos en MoCap, ejecución de la grabación en el NAO y uso del sistema de teleoperación desarrollado.
- Se logró implementar un sistema de teleoperación como base para un sistema más complejo por medio del diseño y uso de diferentes módulos dedicados a tareas específicas.

5.2. Recomendaciones y lecciones aprendidas

El trabajo presente debe ser leído con la intención de entender cómo debe operar un sistema de teleoperación, según la propuesta y diseño seleccionado, y las generalidades de operación del robot humanoide NAO cuando se intenta controlar el movimiento de cuerpo completo. Si se desea emprender un proyecto de desarrollo de un sistema de teleoperación del robot humanoide NAO desde cero (o bien algún otro robot humanoide en que se puedan aplicar las ideas propuestas), el marco teórico pone una buena base para comprender las complicaciones que conlleva el sistema y diferentes consideraciones de operación de un robot humanoide NAO bajo este modelo de control remoto.

Para iniciar un sistema de teleoperación se recomienda conocer las capacidades tanto del sistema que genera la información de control como del sistema operado, ya que el diseño del sistema girará en torno a estas herramientas, y para poder realizar pruebas exitosas, es decir, pruebas que generen información relevante para mejorar la propuesta inicial, ya que si no se sabe cómo plantear una buena prueba, no importa el resultado que se obtenga no se sabrá cómo aprovecharla. Durante el desarrollo del proyecto actual hubo ocasiones en que las pruebas fallaban por no tener certeza de los datos que se estaban utilizando para la ejecución de las pruebas y por no conocer diferentes funciones útiles disponibles para el control del robot humanoide NAO.

También es importante prestar atención a los diferentes términos utilizados, ya que al encontrar estos en algunos de los textos consultados como referencia se pudo orientar mejor la búsqueda de soluciones y llegar, por ejemplo, a conocer dichas funciones útiles disponibles que se desconocían, y que probablemente no se iban a encontrar fácilmente sin esas palabras clave.

Se recomienda además tener bien presente el tiempo disponible y recursos de personal para el desarrollo del proyecto, si se desea que el producto final sea un sistema con la capacidad de operar el robot humanoide NAO con movimientos complejos, se necesitará más tiempo dedicado que si se enfoca los esfuerzos en la operación con movimientos simples. Pero hay que tomar en cuenta que en la generalidad de los proyectos de este estilo, a como se observó con los textos consultados, comenzar con un sistema de teleoperación sencillo y lograr el movimiento de los brazos del NAO es un buen inicio y es

un paso necesario, ya que lo primero que se requiere es tener un sistema que permita la extracción de los datos de control y sea capaz de generar instrucción de ejecución del movimiento en el NAO; sin este sistema funcional no es tan fácil comenzar a hacer pruebas y dedicar esfuerzos en lograr el movimiento complejo del cuerpo completo del robot NAO, además que el proyecto se extendería mucho para lograr una propuesta así de ambiciosa.

En cuanto al sistema creado, aunque se haya logrado teleoperar el robot NAO y adquirir el conocimiento obtenido sobre el funcionamiento tanto del robot NAO como del MoCap, este sistema aún no satisface los requerimientos de aquel que se necesita para los diferentes proyectos en los que se quiere introducir (revisar la sección 1.2), y por ello es importante los esfuerzos siguientes en mejorar el funcionamiento del sistema y de añadir mayores capacidades.

Es importante mencionar que el sistema presentado está estructurado en módulos que realizan tareas específicas, de modo que para mejorar las capacidades del sistema actual se pueda trabajar de manera independiente en la creación de módulos con las nuevas capacidades deseadas, como un módulo de extracción de datos en *tiempo real* y uno para el balance de movimientos complejos, requiriendo pequeñas modificaciones en los principales *Calibration.py* y *Move.py*.

5.2.1. Consideraciones sobre el sistema de teleoperación desarrollado

El sistema de teleoperación desarrollado tiene mucho camino por recorrer aún, con grandes mejoras que se le pueden hacer. Permite la ejecución de movimientos simples para el robot humanoide NAO, esto es el movimiento de ambos brazos, con un módulo de calibración por regresión lineal de primer y segundo orden, con un control de balance del NAO muy básico.

Se debe realizar pruebas con diferentes algoritmos de ajuste de curvas para el proceso de calibración que permita curvas más suaves y efectividad en la calibración para todos los DoF disponibles para cada actuador involucrado. Además, por diferentes problemas técnicos a la hora de crear las rutinas de calibración, no fue posible generar información para el movimiento de la cabeza del robot NAO, de modo que no se realiza la calibración de la cadena **Head**, lo cual puede afectar el balance. También se debe considerar que para la calibración de los datos se debe realizar un proceso manual de revisión y acople de estos en la linea temporal para que la calibración sea efectiva, así como de lograr que la ejecución de la persona en las rutinas de calibración sean lo más parecida a la ejecución del robot NAO, para que los datos sean comparables.

Es necesaria la creación de un módulo de balance del NAO dedicado al análisis de los datos de ubicación espacial que se le pasarán al robot para permitir ajustar las restricciones y condiciones de balance, así como la reubicación del **Torso** y el COM para permitir un mejor balance dinámico y permitir la ejecución de movimientos complejos, esto es el movimiento de las piernas y el torso.

Además, se debe recordar que el sistema de teleoperación actual utiliza grabaciones (datos previamente exportados y guardados) de la captura de movimiento de la persona, hace falta un módulo que permita la obtención y procesamiento de estos datos en *tiempo real*.

5.2.2. Propuesta de mejoras del sistema de teleoperación

Según las consideraciones anteriores sobre el sistema de teleoperación del presente trabajo se recomienda enfocarse en las siguientes ideas para mejorar las capacidades del sistema y permitir un mejor control y ejecución del movimiento por el robot humanoide NAO.

- Implementar un método de detección de patrones para acoplar sobre la línea temporal los datos de la grabación de la persona correspondientes a los del NAO, para la calibración.
- Encontrar un mejor ajuste de curvas para la calibración.
- Desarrollar un módulo que permita la teleoperación en *tiempo real*.
- Desarrollar un módulo dedicado al balance del robot humanoide NAO mientras está en movimiento que permita al robot humanoide NAO ejecutar movimientos complejos y mejorar la ejecución de los movimientos simples.
- Incluir la capacidad de desplazamiento del robot humanoide NAO en la teleoperación.
- Entrenar a la persona a ejecutar los movimientos para que la calibración pueda utilizar datos comparables en la intención del movimiento con respecto a los datos del robot NAO.

Además de estas propuestas, y otras que puedan surgir, se ha tenido la idea de unir este sistema de teleoperación con un sistema de telepresencia, de modo que se genere un sistema más complejo en que por medio de las grabaciones del MoCap (o bien en *tiempo real* en una versión futura) se controle el movimiento del cuerpo completo del NAO en cuanto a extremidades se refiere, y por el sistema de telepresencia se logre compartir la percepción del robot NAO (por ejemplo de las cámaras frontales) con la persona teleoperando, inclusive que se logre controlar el desplazamiento en el espacio del NAO por medio de algún otro sistema de captura más local, como de captura de gestos con las manos, que no es requerido capturar por el MoCap para la teleoperación.

Apéndice A

Resultados completos de la validación del sistema de teleoperación

A continuación se presentan todas las imágenes y gráficas resultado del proceso de validación del sistema de teleoperación en las secciones A.1 y A.2 respectivamente. Las imágenes corresponden a una comparación visual entre el movimiento efectuado por una persona y el movimiento ejecutado por el robot humanoide NAO al intentar imitar dicho movimiento. Las gráficas son utilizadas como parte de una comparación analítica entre los datos obtenidos del MoCap, los datos de los sensores en los actuadores del robot NAO y los datos obtenidos por el proceso de calibración.

Para la validación se utilizan las grabaciones *Rutina_Círculos_1* y *Círculos-con-Lateral* para la comparación visual, cuyo contenido a ejecutar se muestra en las imágenes de la ejecución por la persona con las posiciones principales. Ambas comienzan con la posición **Stand** e incluyen movimiento en círculos para los brazos, ambos en direcciones opuestas. Se tomó datos de grabaciones ejecutadas por dos personas diferentes (Sujeto1 y Sujeto2). Para la comparación gráfica se usan las rutinas de calibración.

A.1. Intención de movimiento

A.2. Comparación gráfica del ajuste de los datos

Para el ajuste de los datos realizado en la calibración se utilizó la regresión lineal para polinomios de grado 1 y 2. Experimentalmente funcionó mejor el uso de polinomio de grado 1. En general se puede observar que los peores ajustes se dan en las rotaciones, esto debido a la mala relación existente entre la capacidad del robot humanoide NAO de controlar la rotación de sus actuadores con la del ser humano, que involuntariamente puede provocar movimientos que *ensucien* la toma de datos.

A.2.1. Polinomio grado 1

RArm

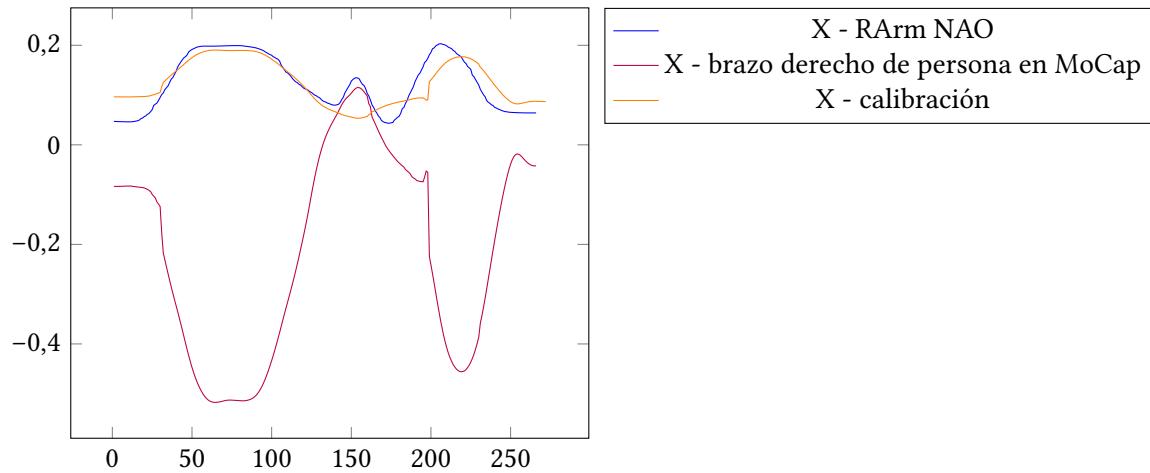


Figura A.1: Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

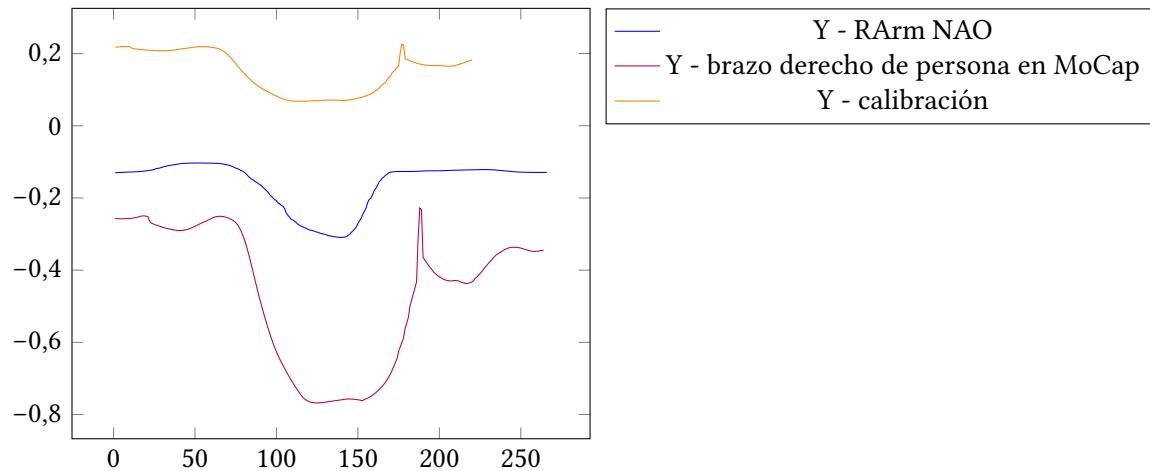


Figura A.2: Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

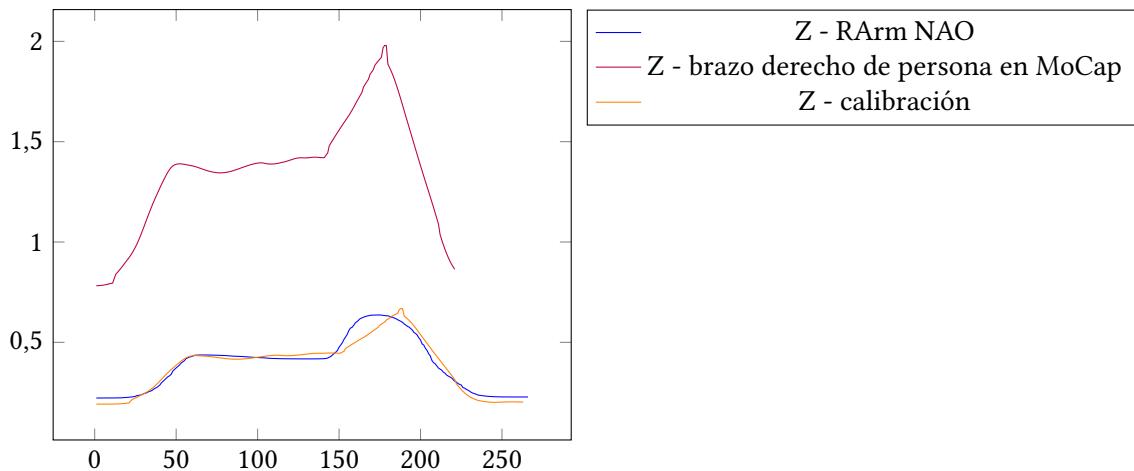


Figura A.3: Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

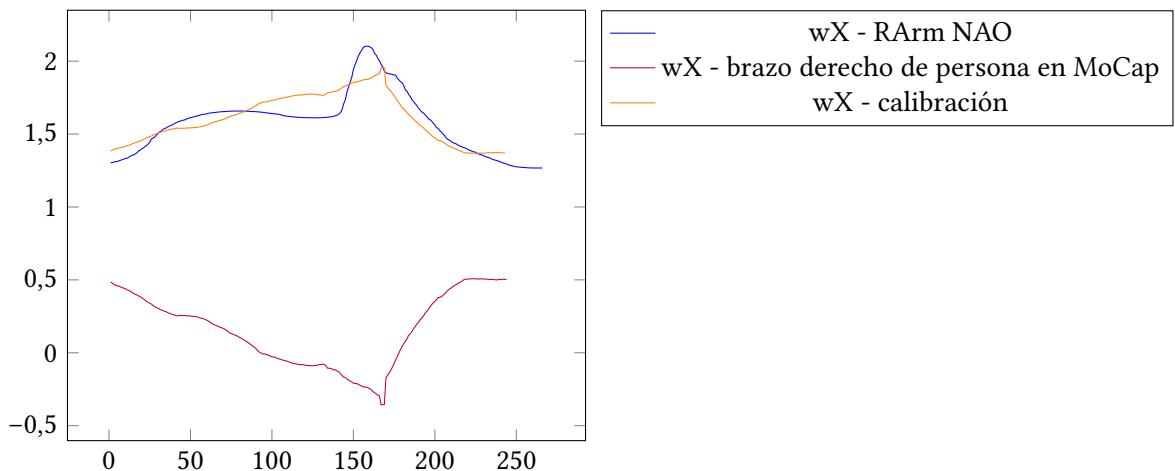


Figura A.4: Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

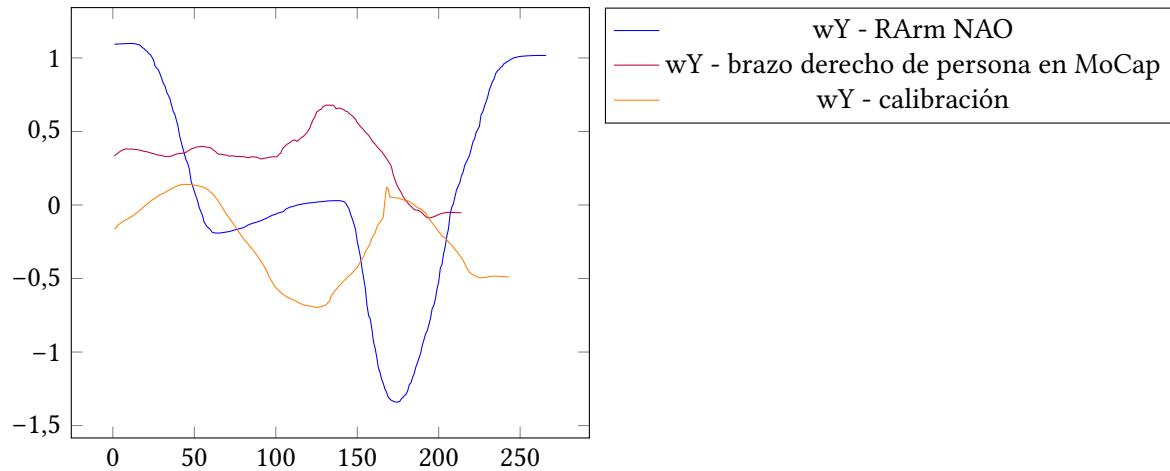


Figura A.5: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

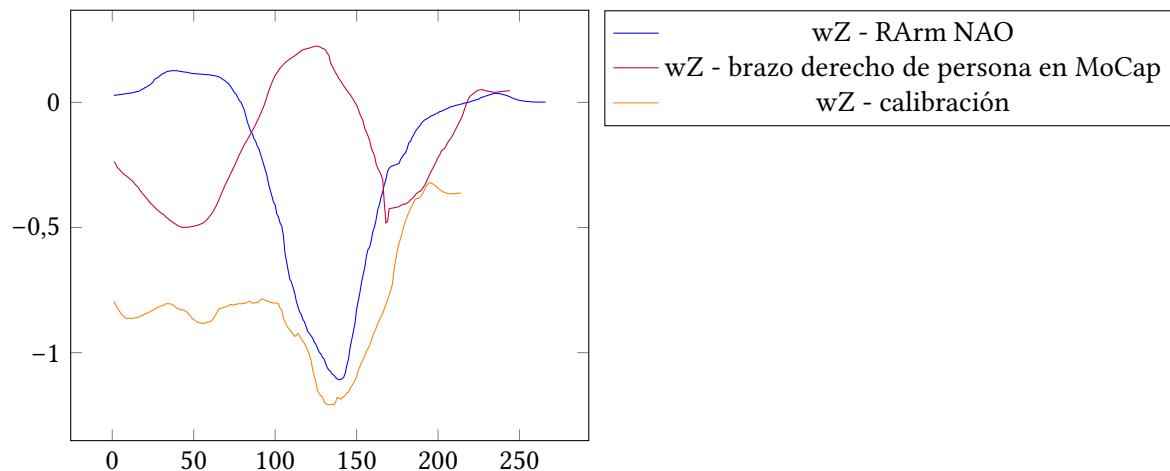


Figura A.6: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

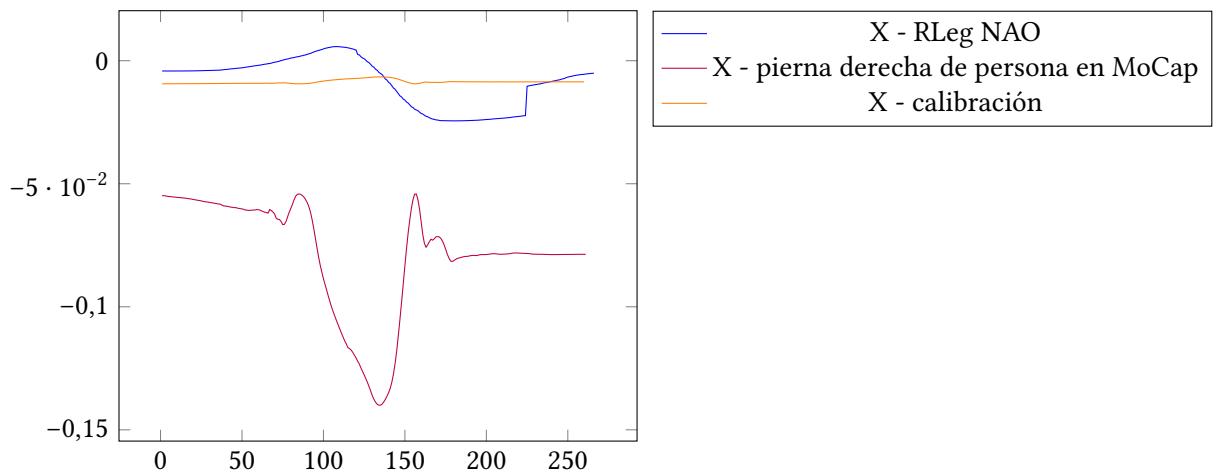
RLeg

Figura A.7: Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

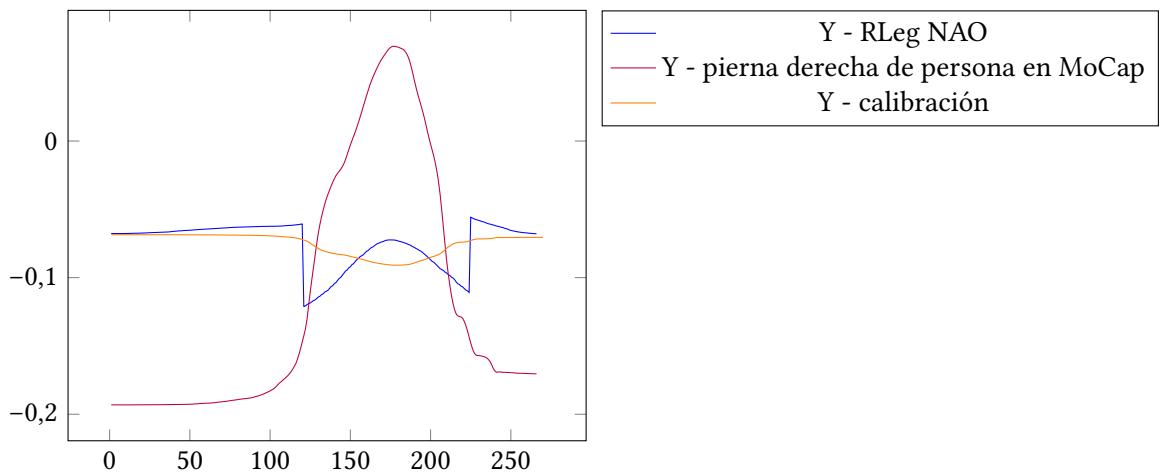


Figura A.8: Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

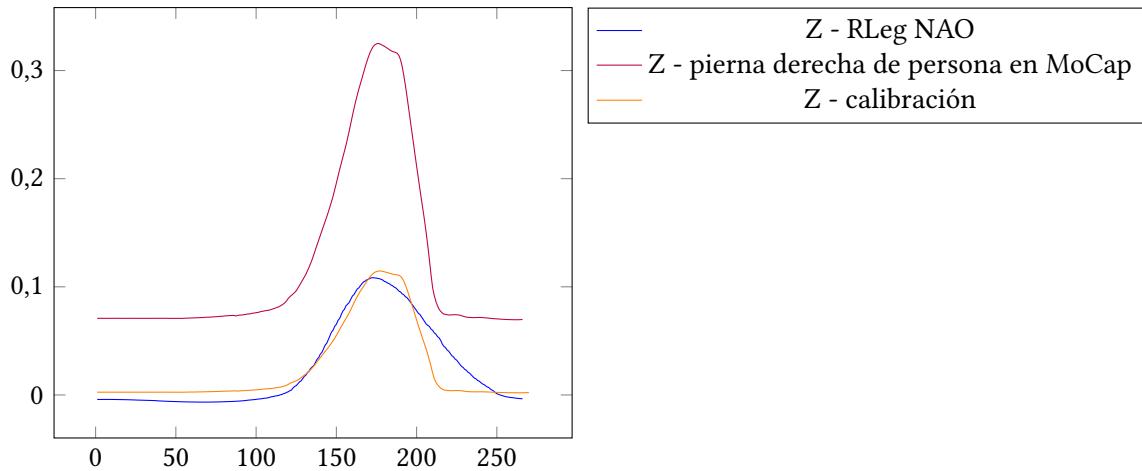


Figura A.9: Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

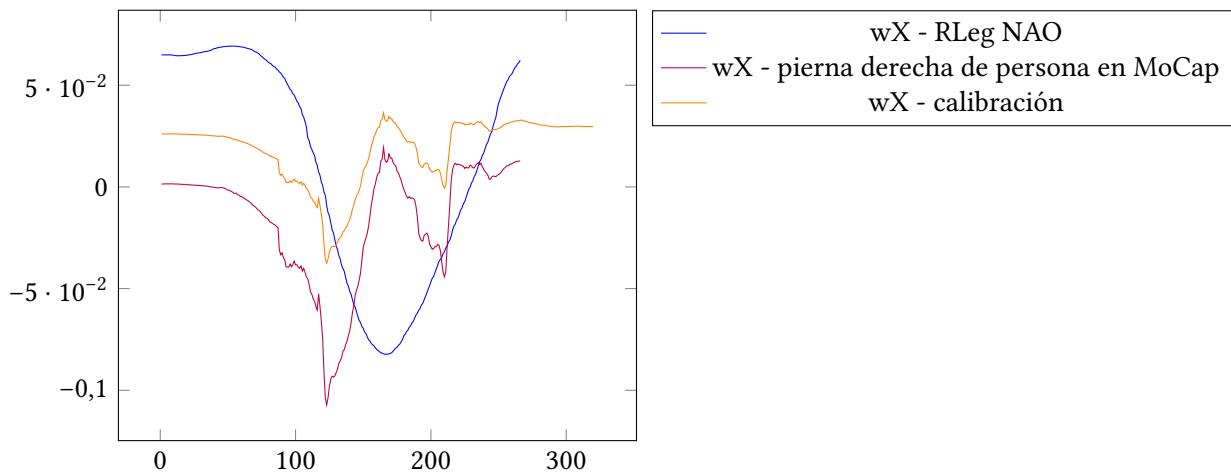


Figura A.10: Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

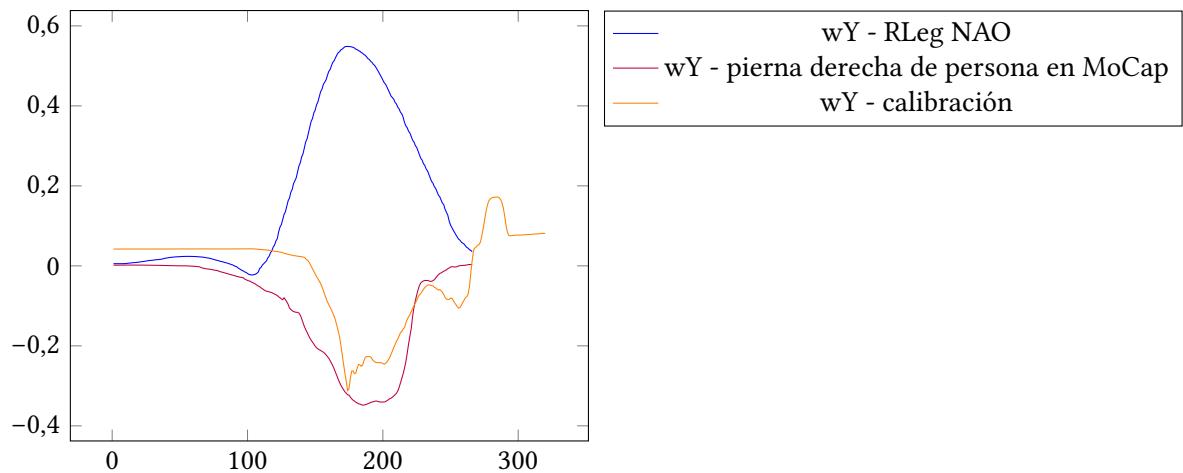


Figura A.11: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

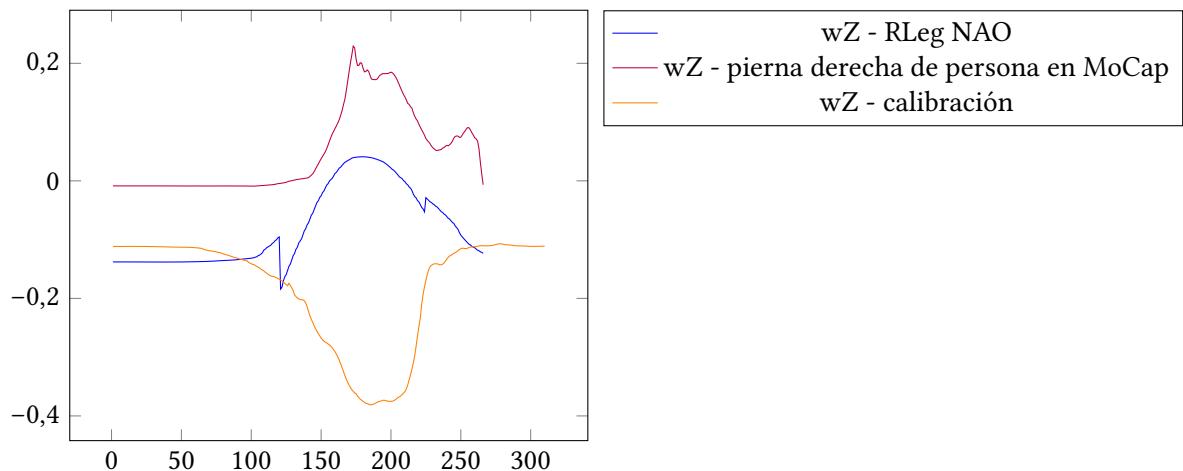


Figura A.12: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

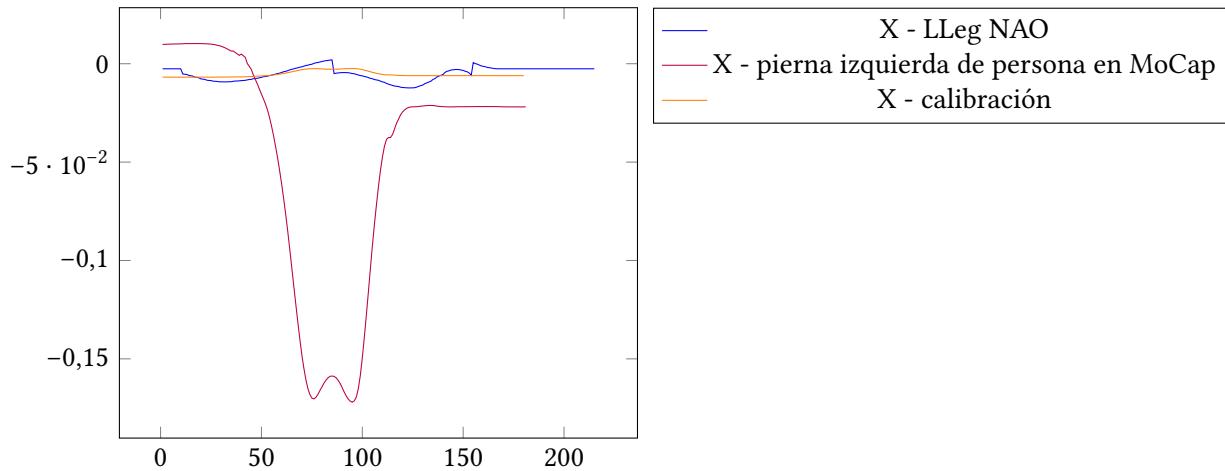
LLeg

Figura A.13: Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

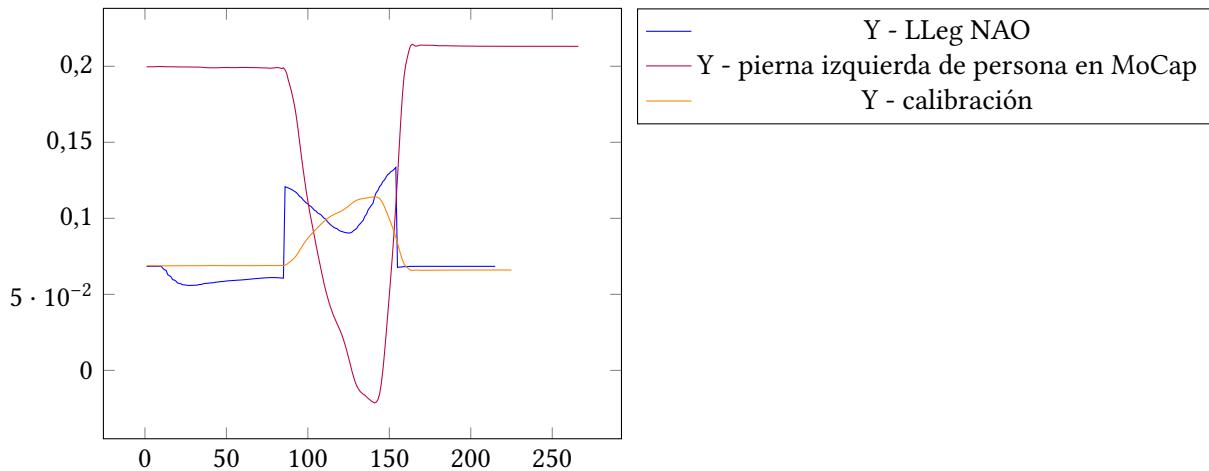


Figura A.14: Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

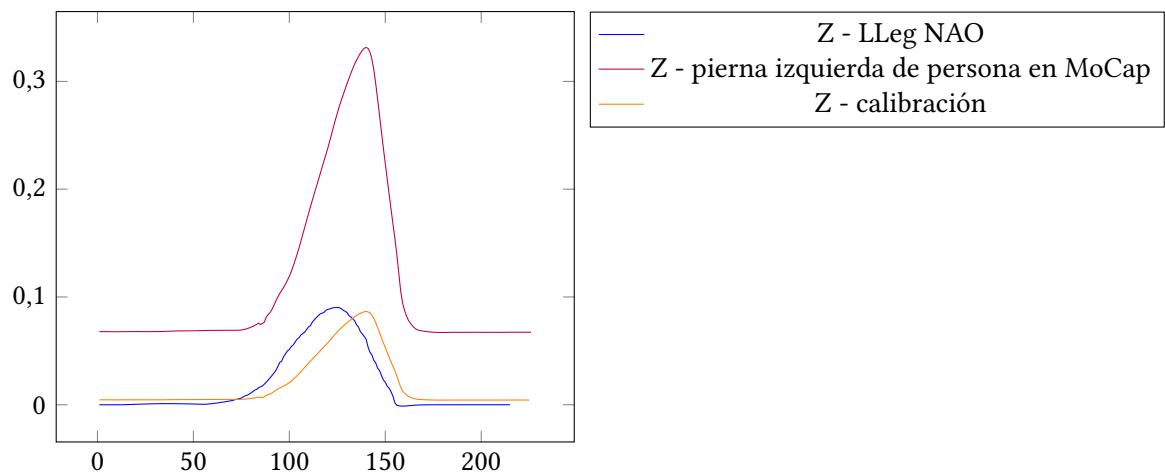


Figura A.15: Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

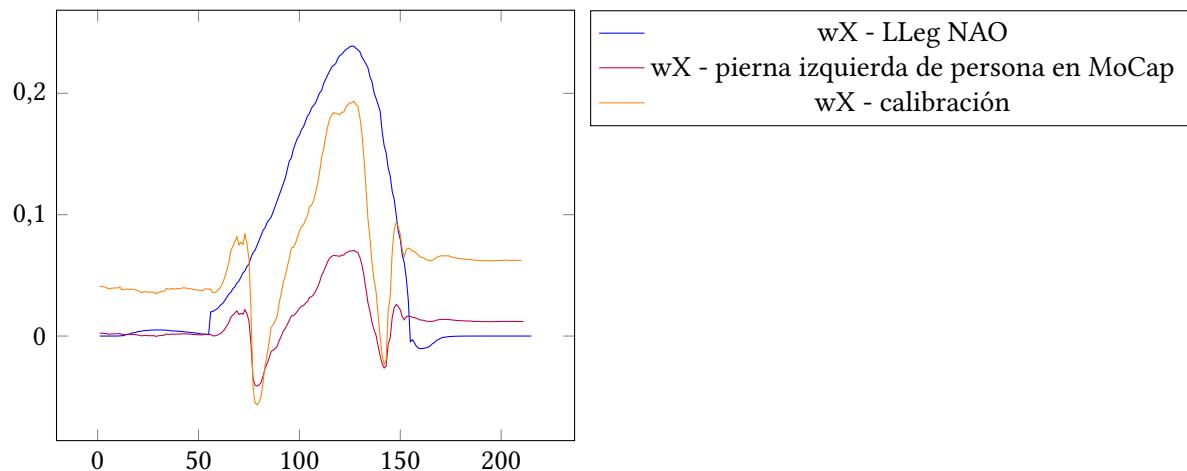


Figura A.16: Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

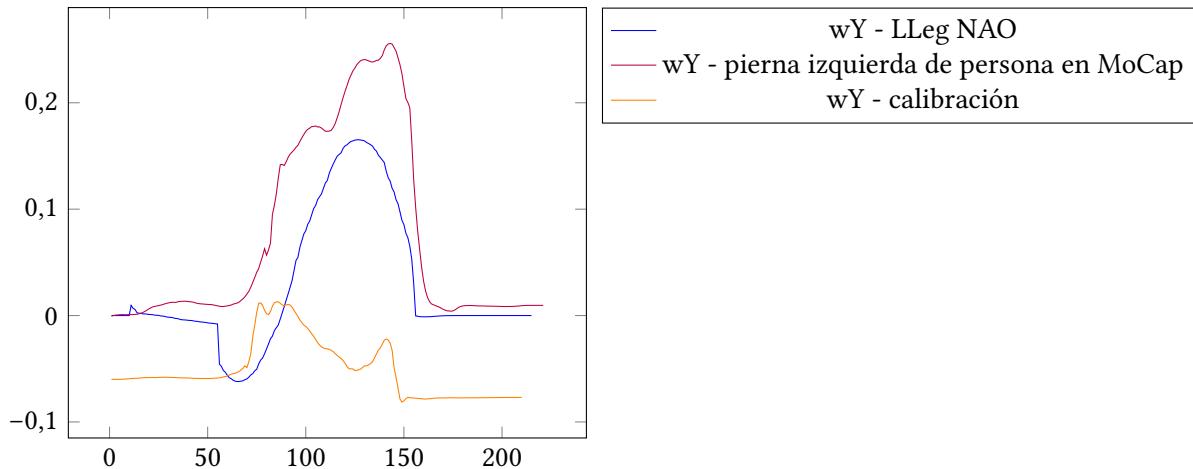


Figura A.17: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

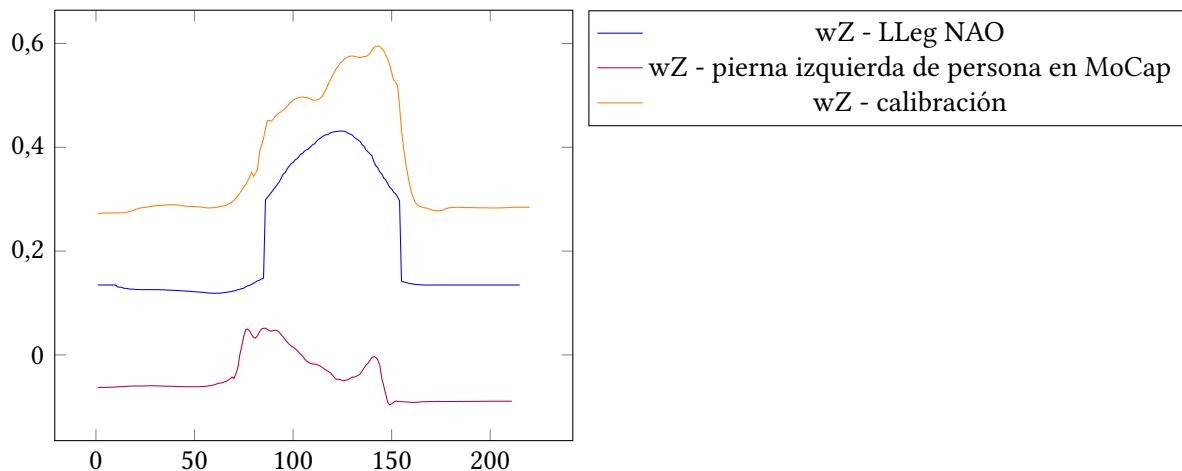


Figura A.18: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

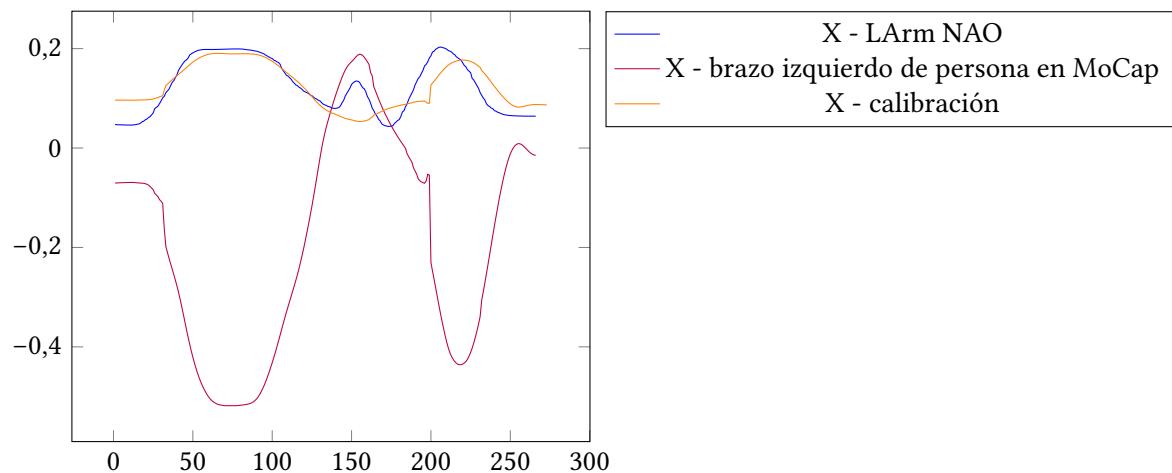
LArm

Figura A.19: Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

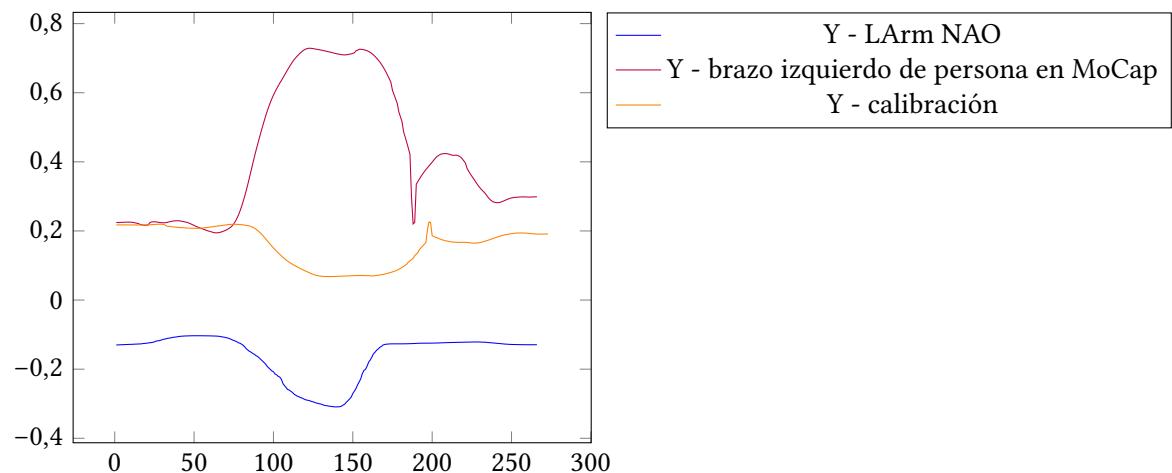


Figura A.20: Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

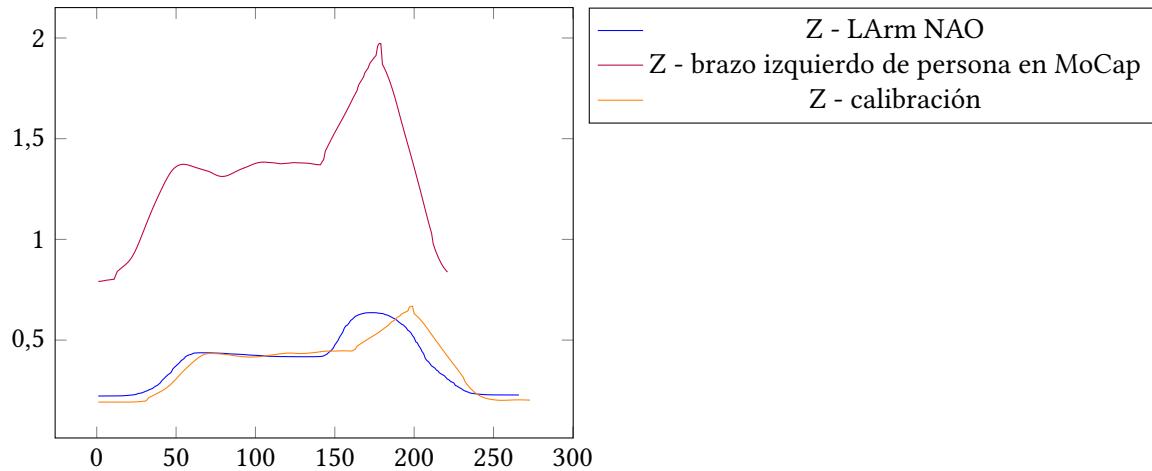


Figura A.21: Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

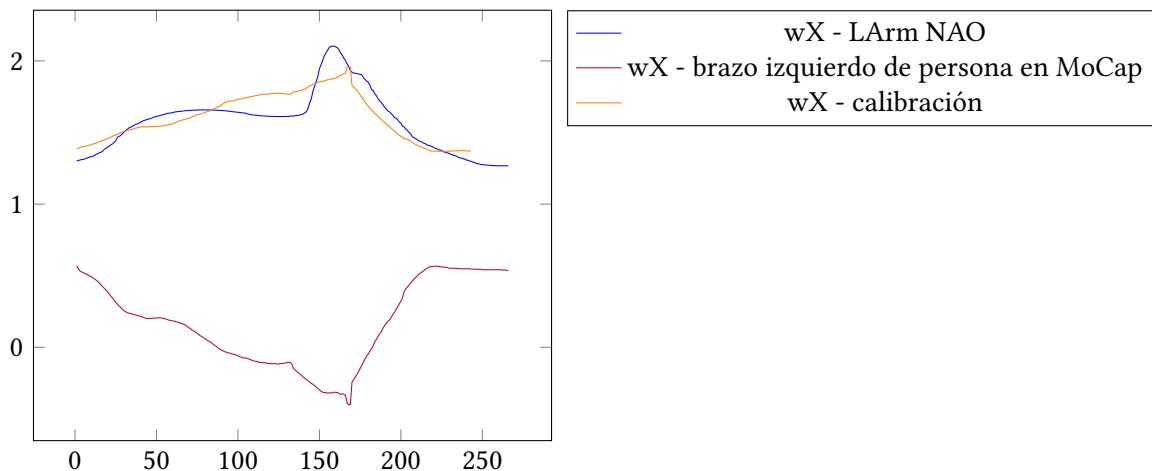


Figura A.22: Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

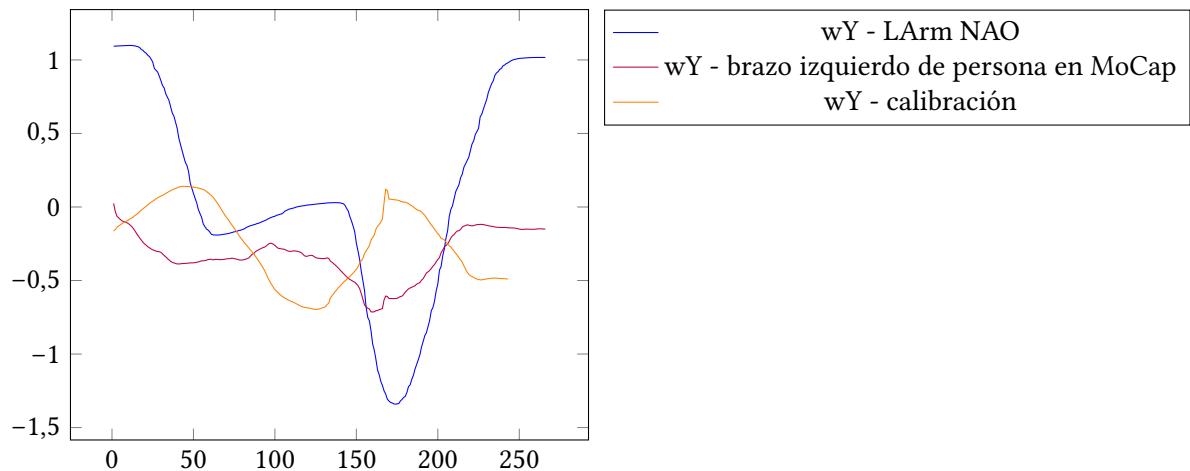


Figura A.23: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

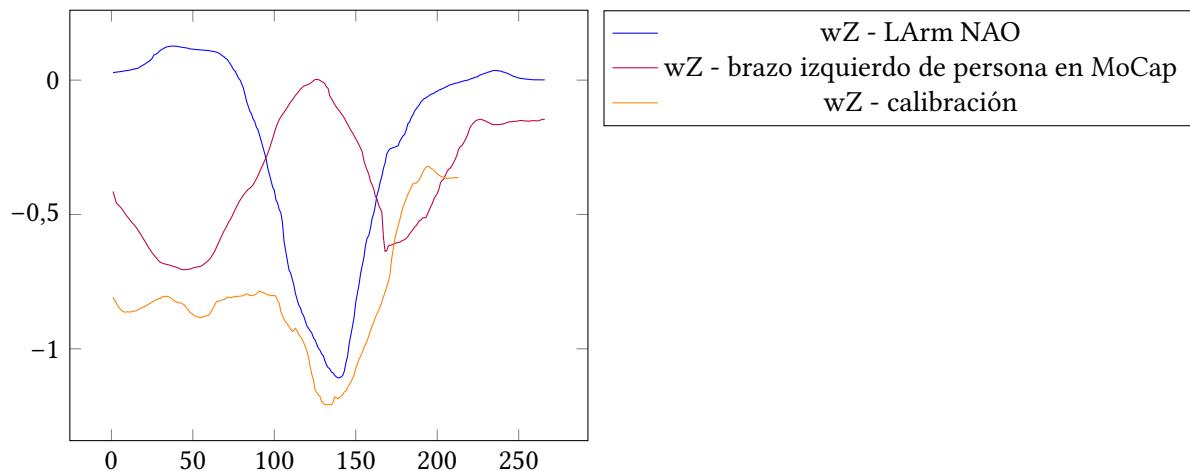


Figura A.24: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

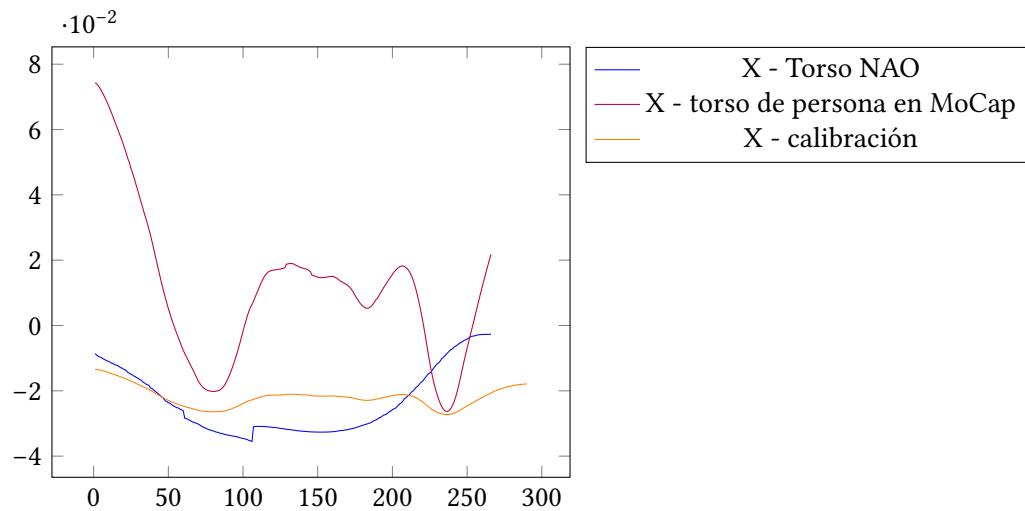
Torso

Figura A.25: Ajuste polinomio grado 1 en datos del eje X para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

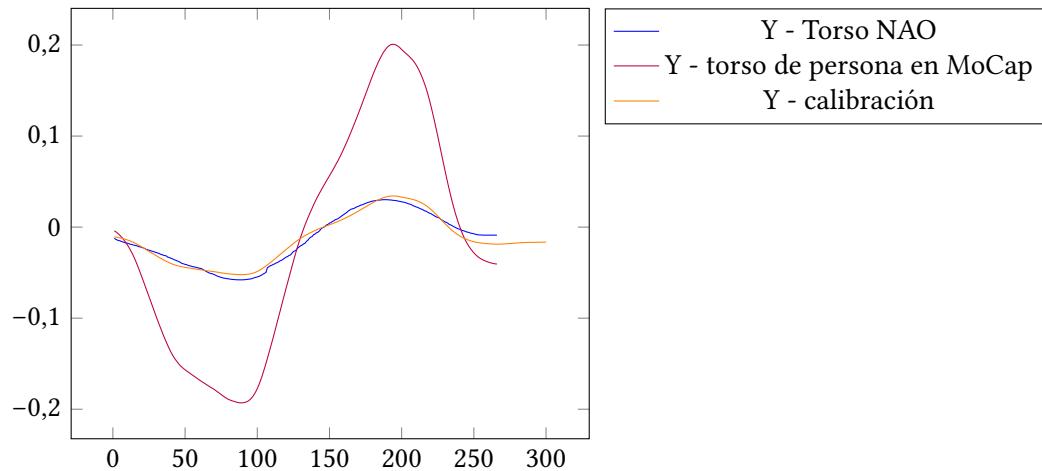


Figura A.26: Ajuste polinomio grado 1 en datos del eje Y para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

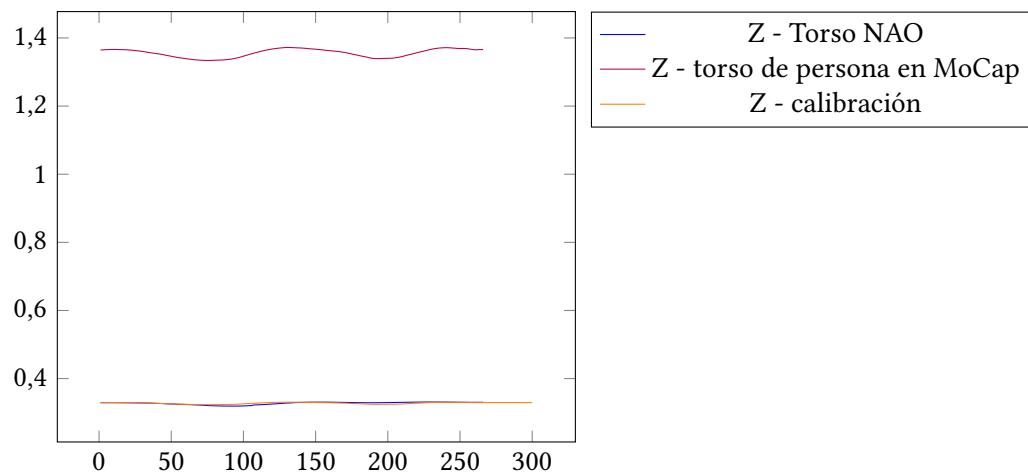


Figura A.27: Ajuste polinomio grado 1 en datos del eje Z para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

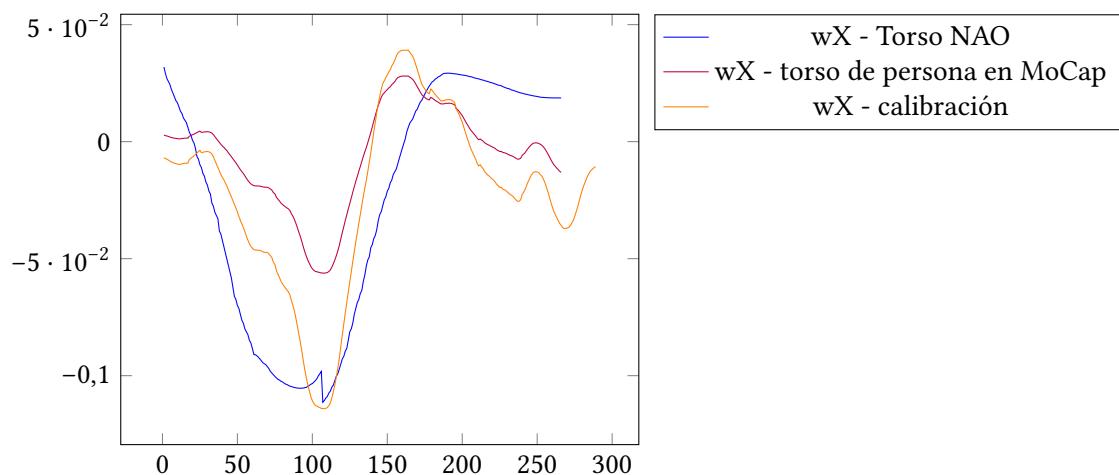


Figura A.28: Ajuste polinomio grado 1 en datos de rotación alrededor del eje X para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

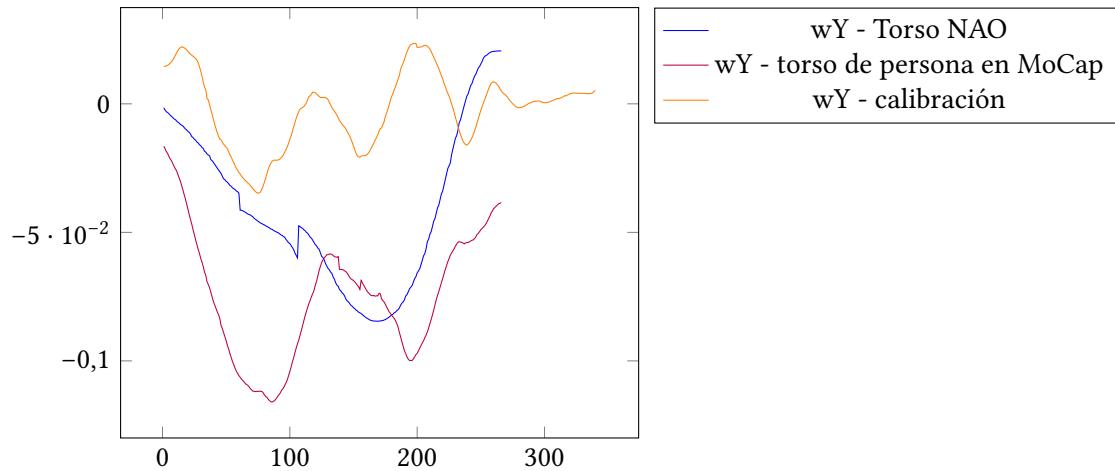


Figura A.29: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Y para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

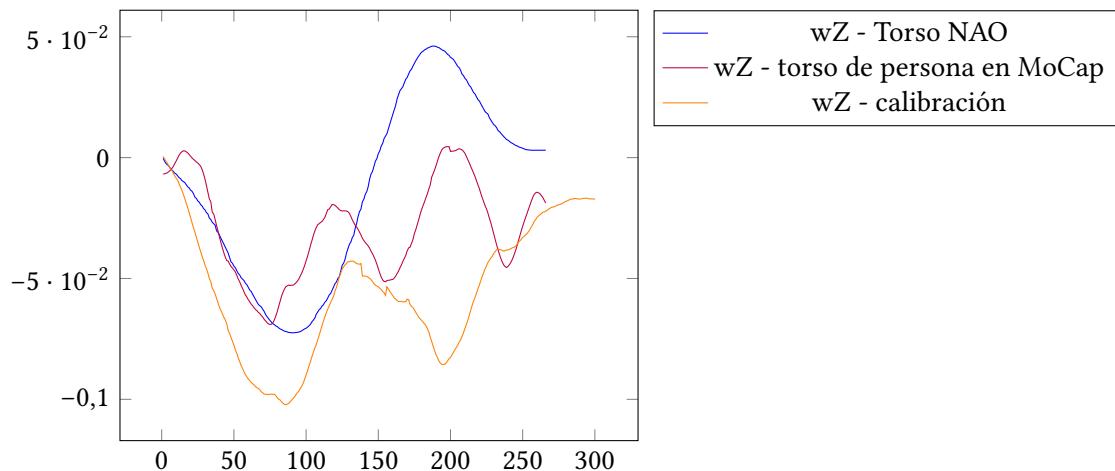


Figura A.30: Ajuste polinomio grado 1 en datos de rotación alrededor del eje Z para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A.2.2. Polinomio grado 2

RArm

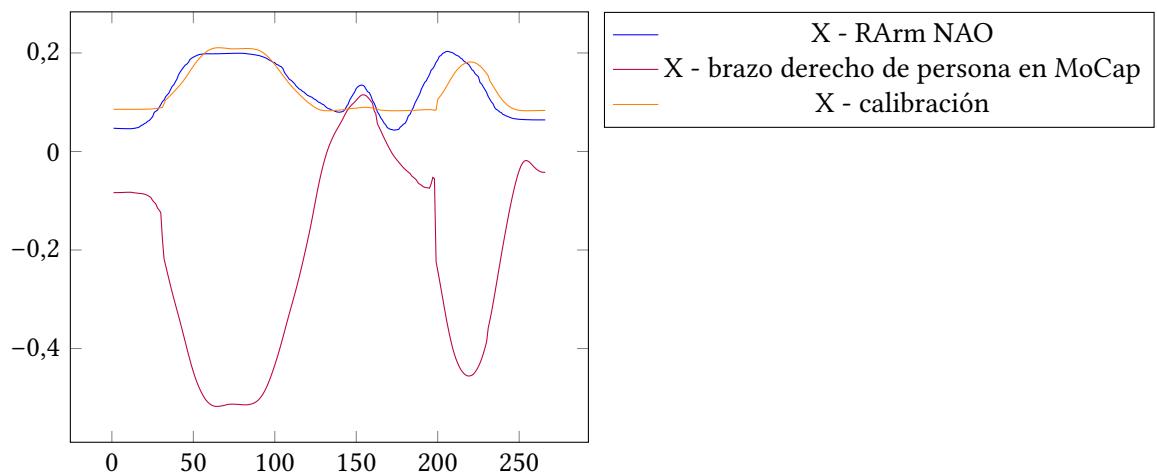


Figura A.31: Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

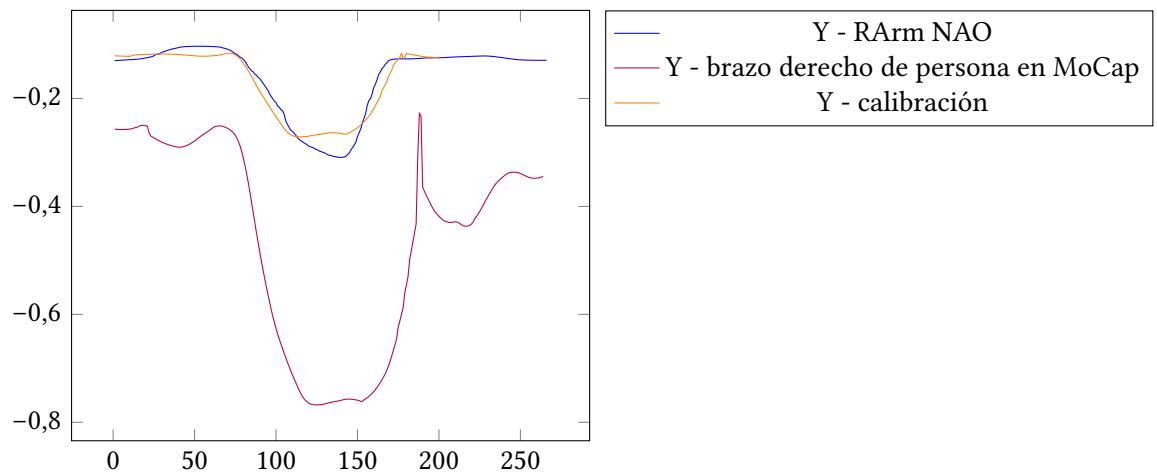


Figura A.32: Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

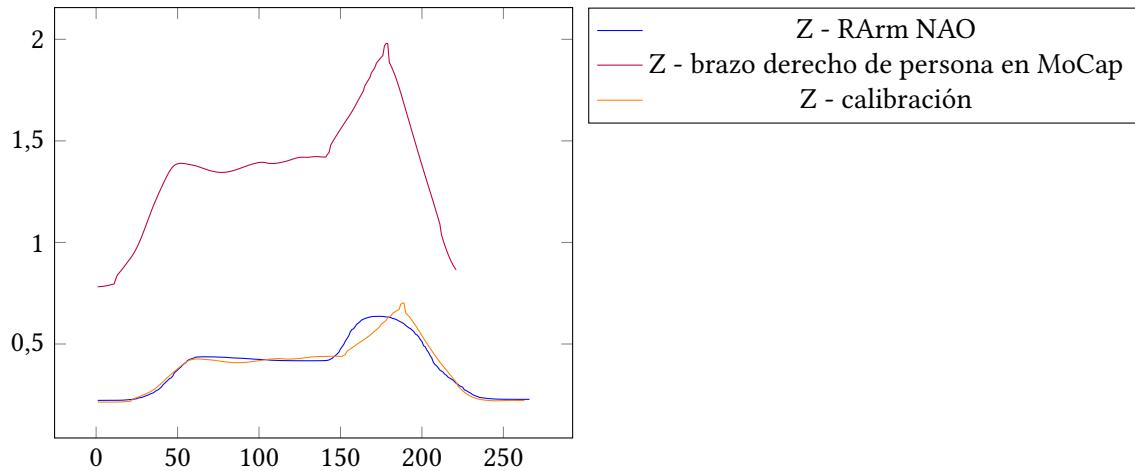


Figura A.33: Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

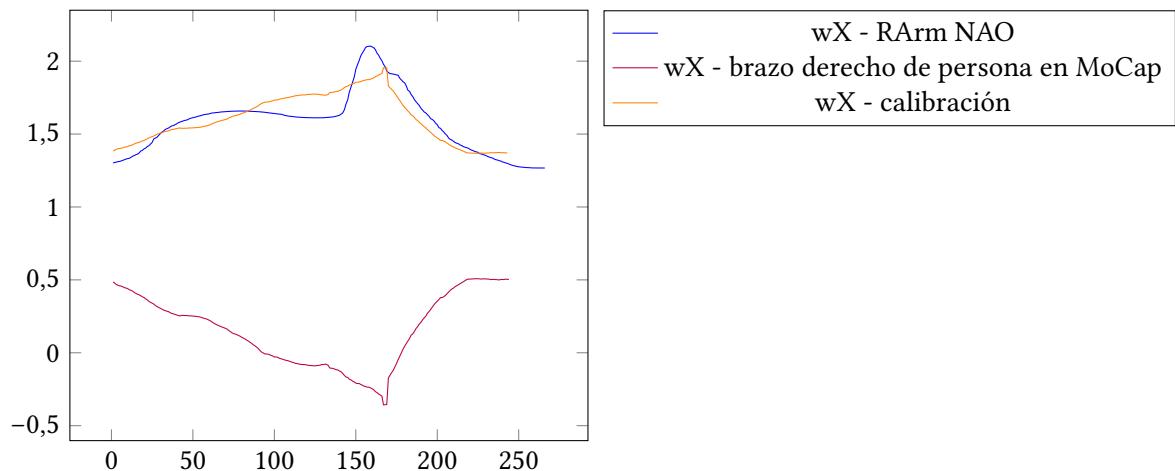


Figura A.34: Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

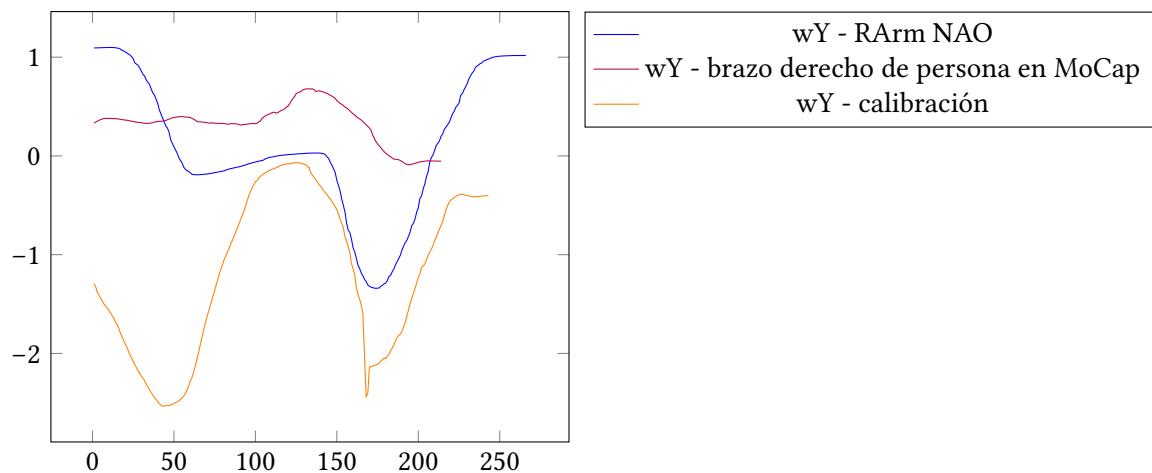


Figura A.35: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

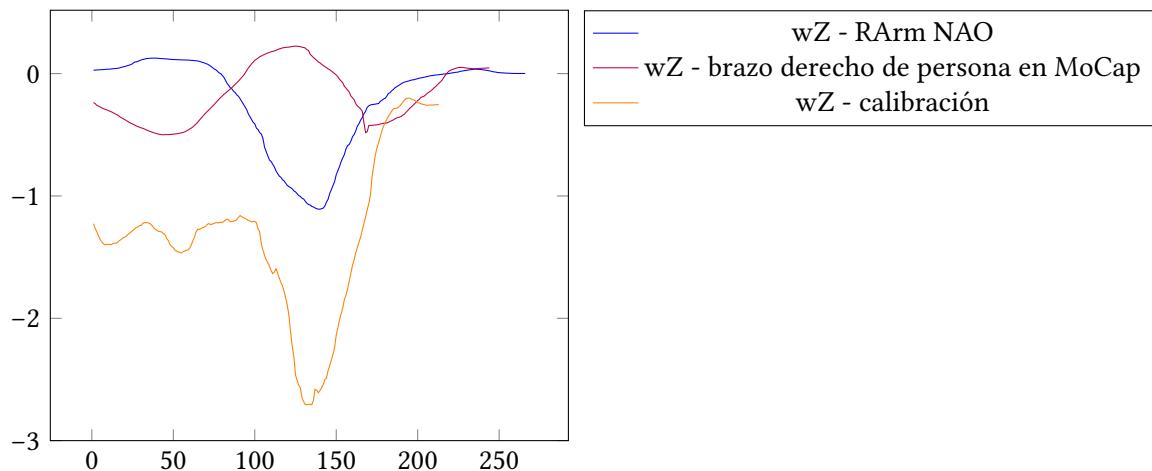


Figura A.36: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores RArm del robot humanoide NAO (brazo derecho). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

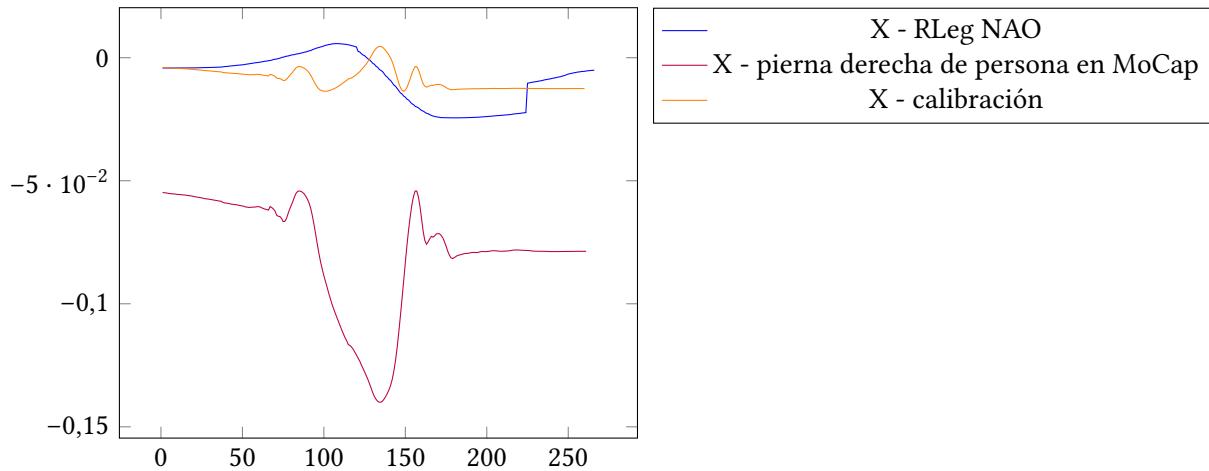
RLeg

Figura A.37: Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

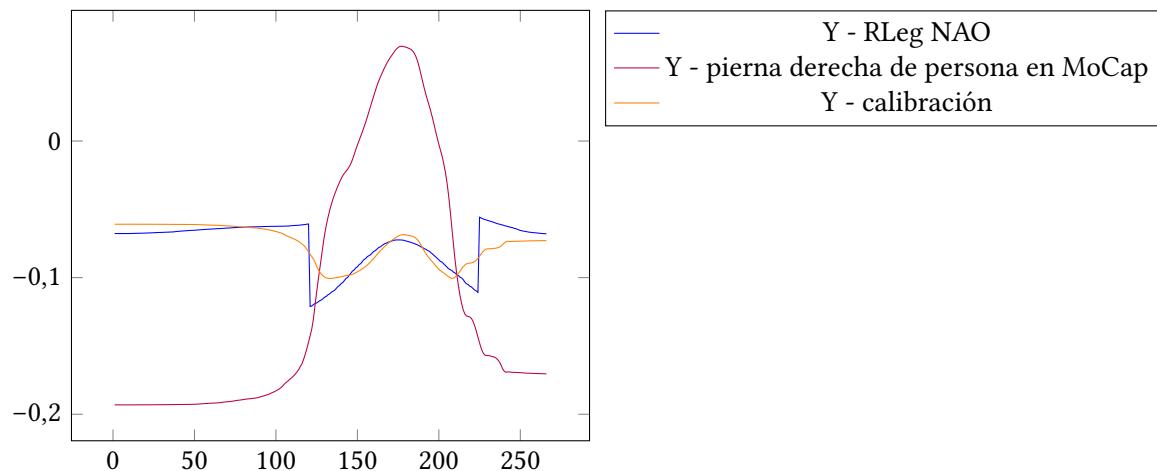


Figura A.38: Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

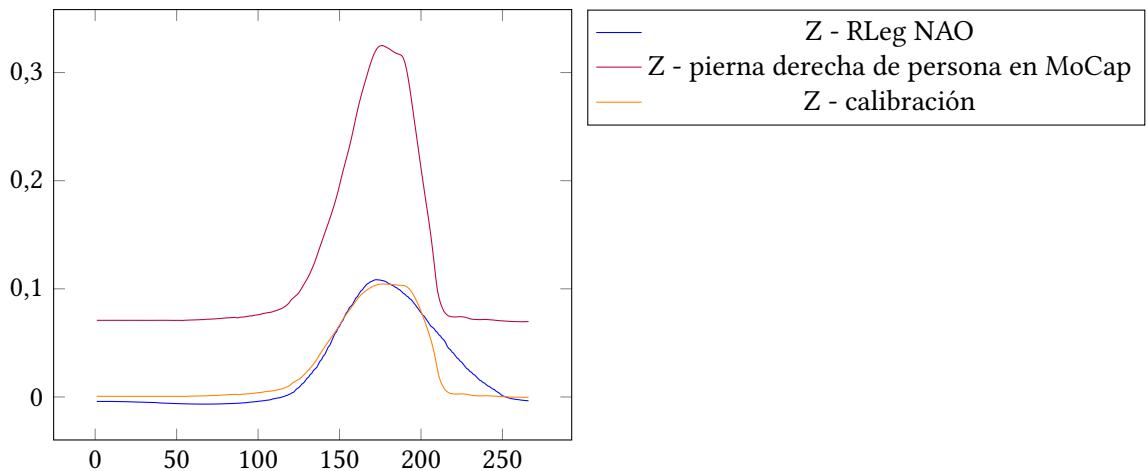


Figura A.39: Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

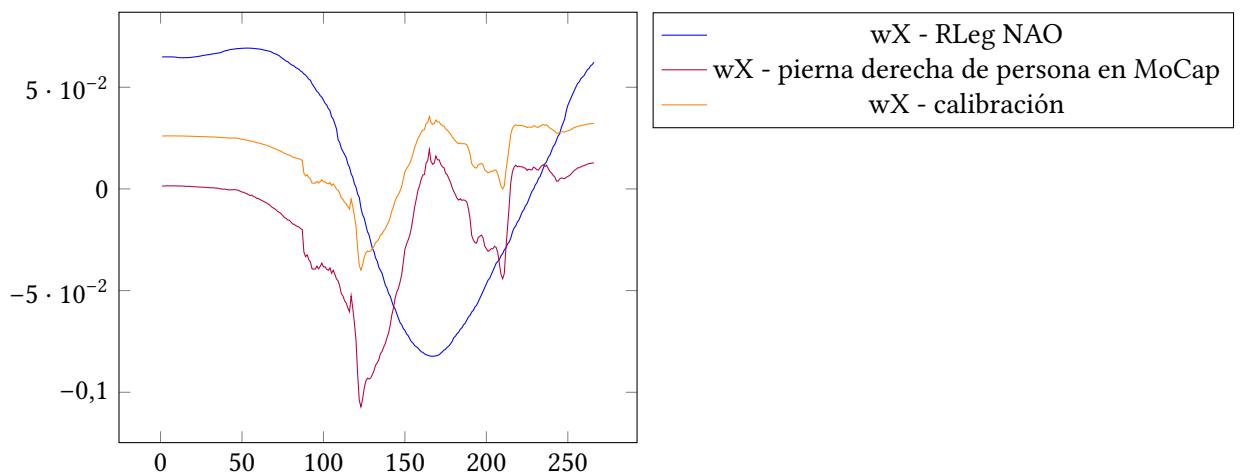


Figura A.40: Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

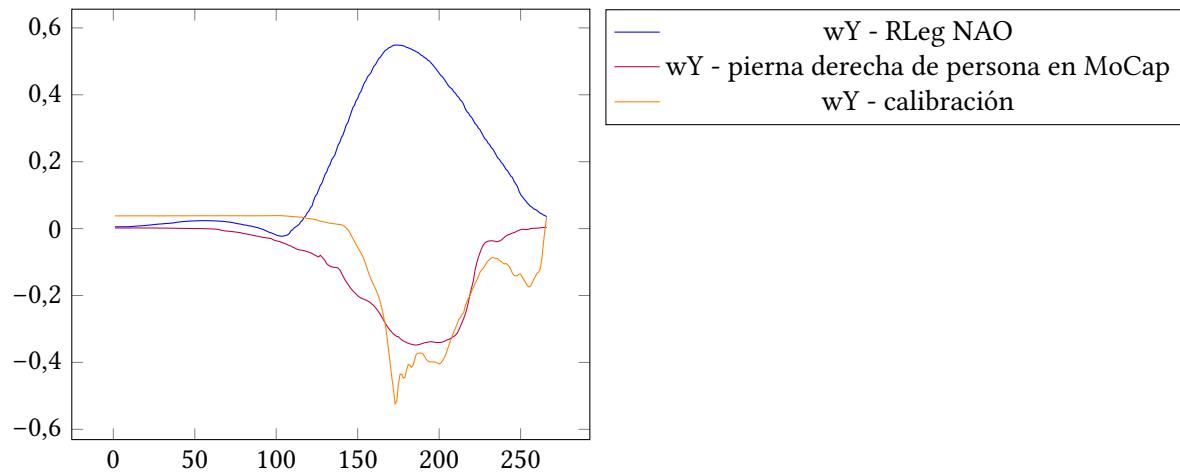


Figura A.41: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

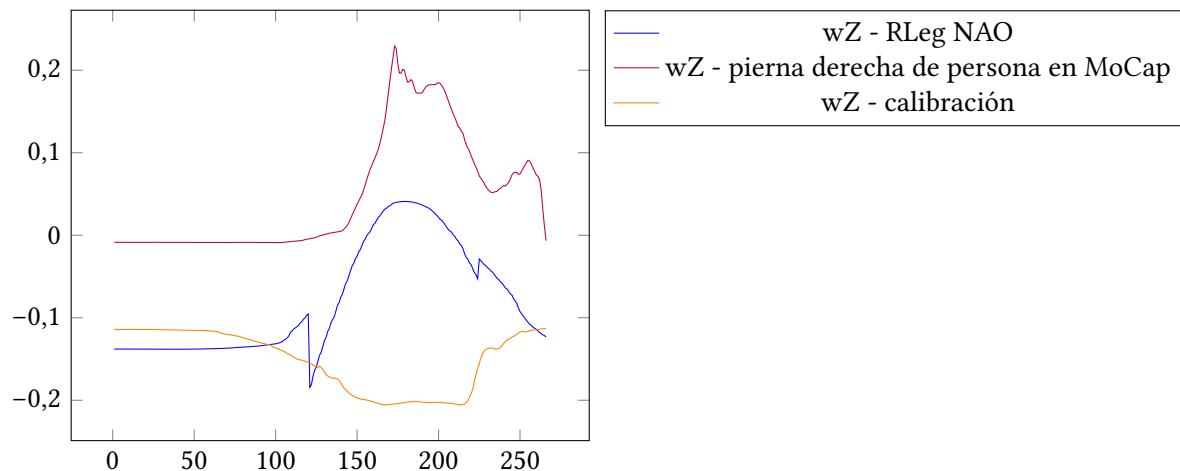


Figura A.42: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores RLeg del robot humanoide NAO (pierna derecha). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

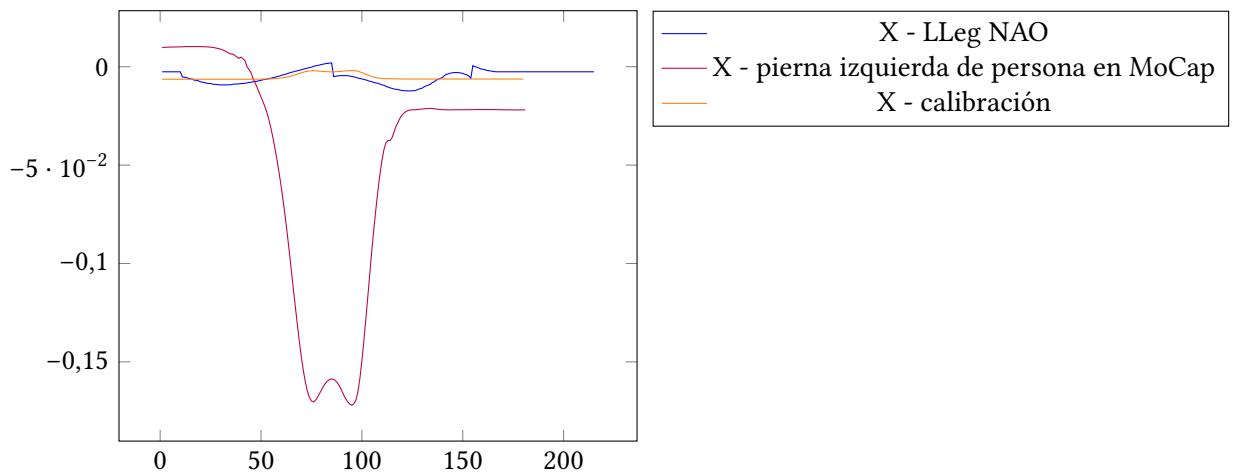
LLeg

Figura A.43: Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

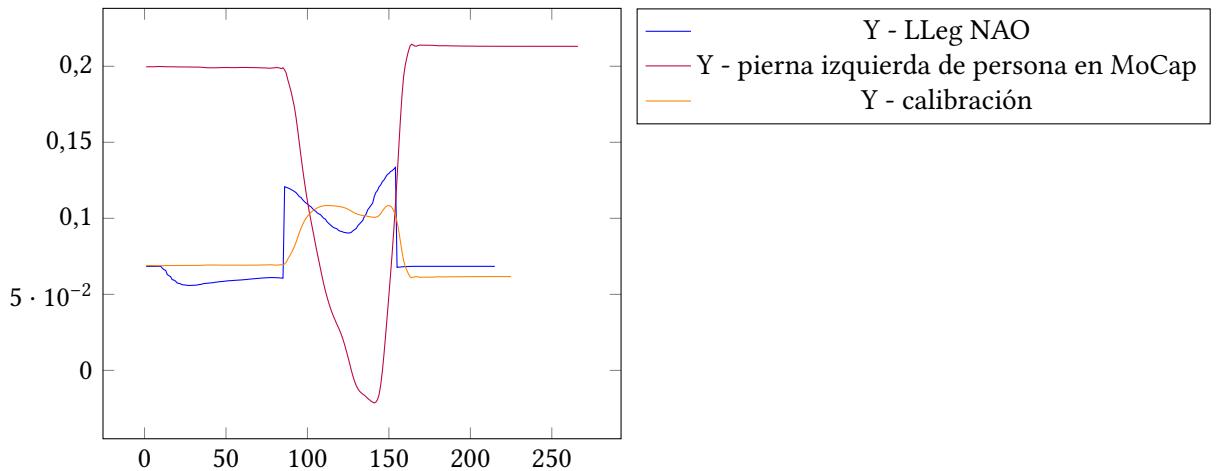


Figura A.44: Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

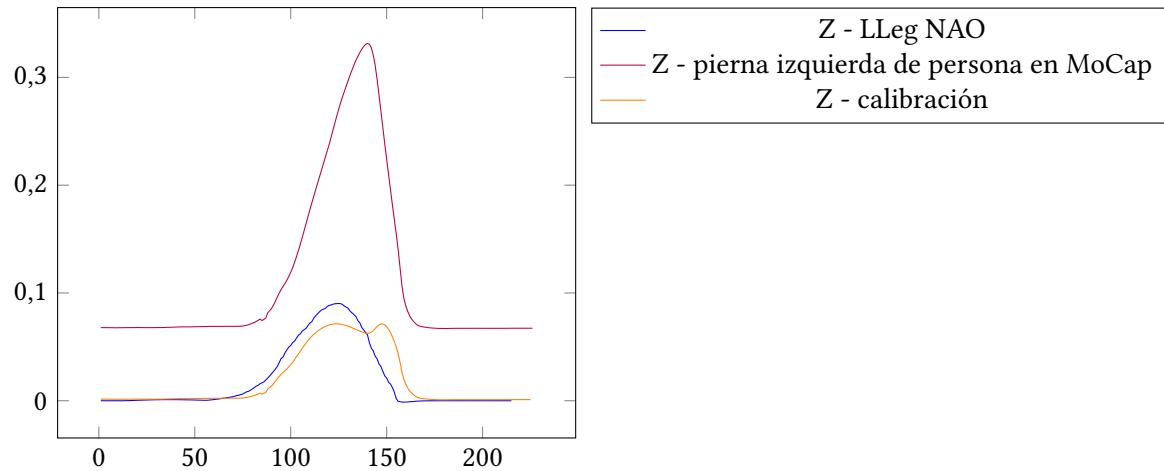


Figura A.45: Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

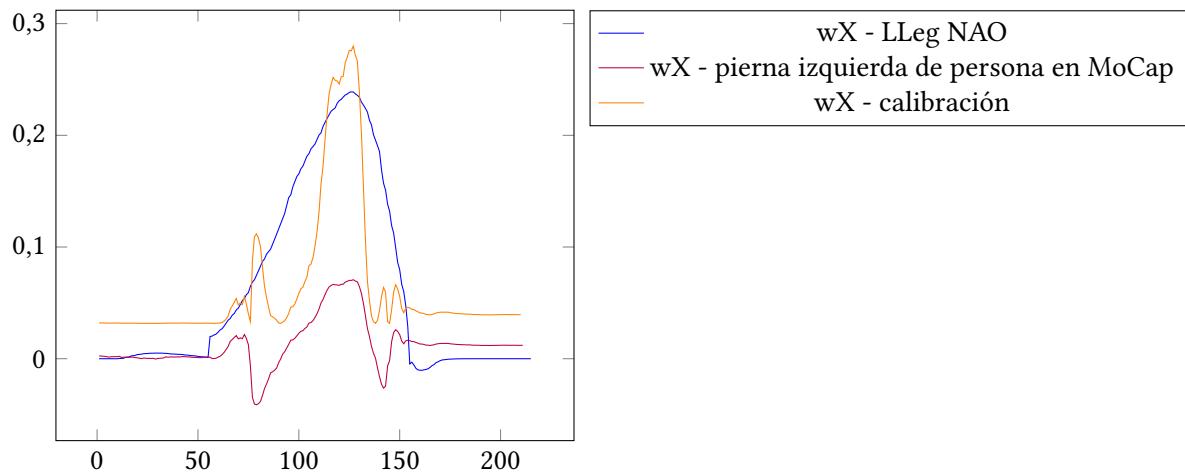


Figura A.46: Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

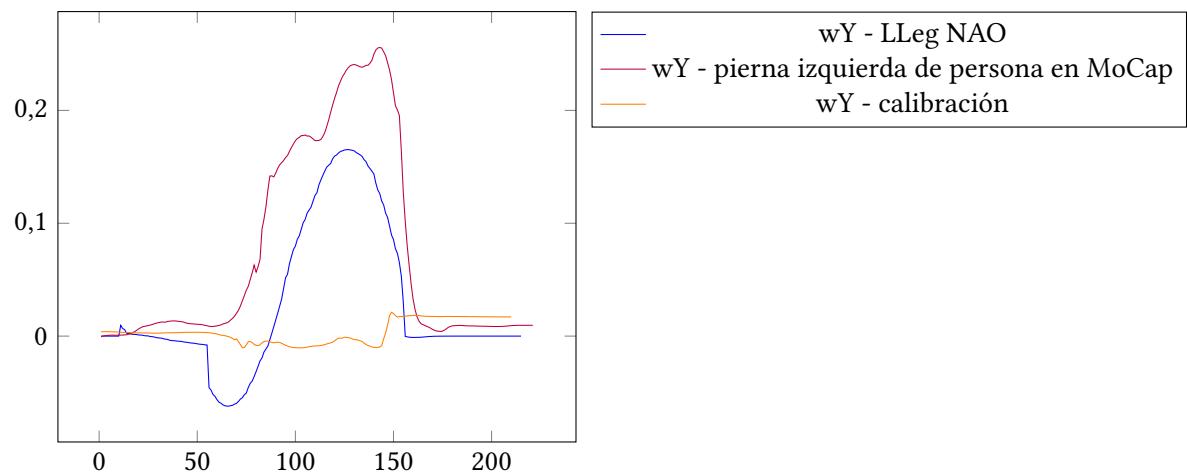


Figura A.47: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

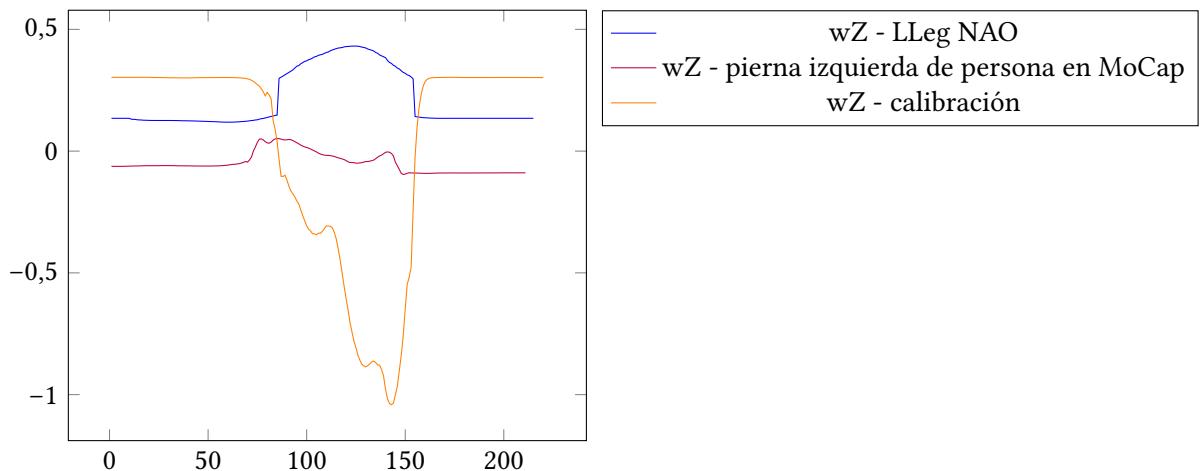


Figura A.48: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores LLeg del robot humanoide NAO (pierna izquierda). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

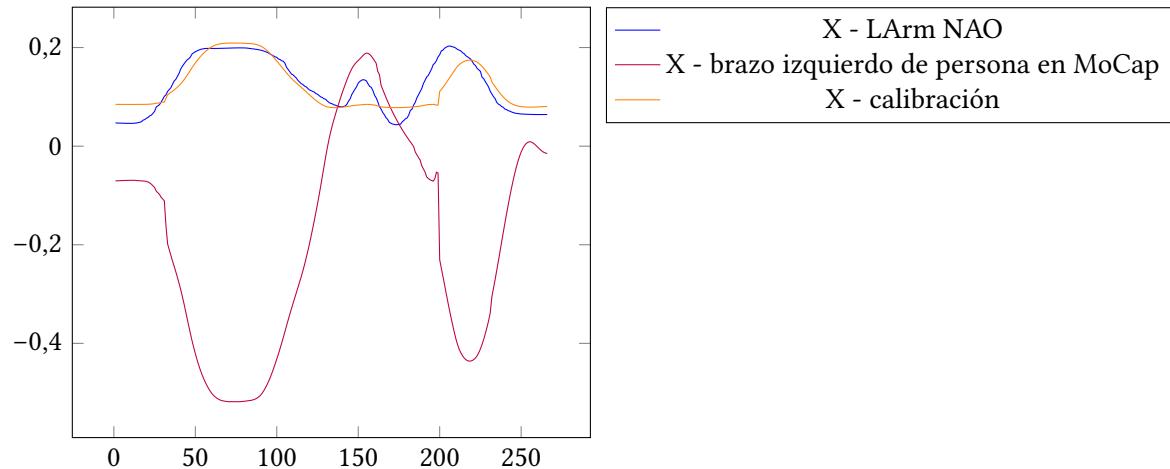
LArm

Figura A.49: Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

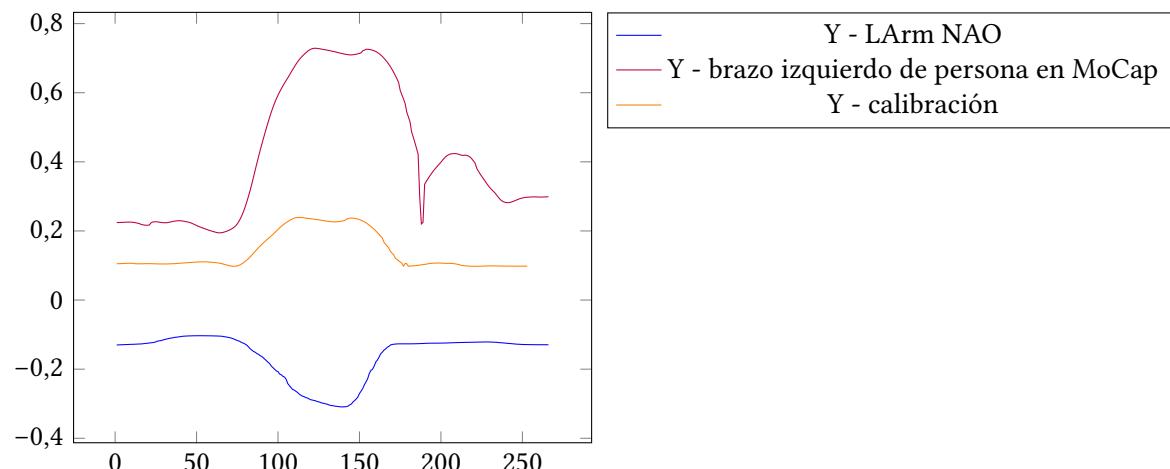


Figura A.50: Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

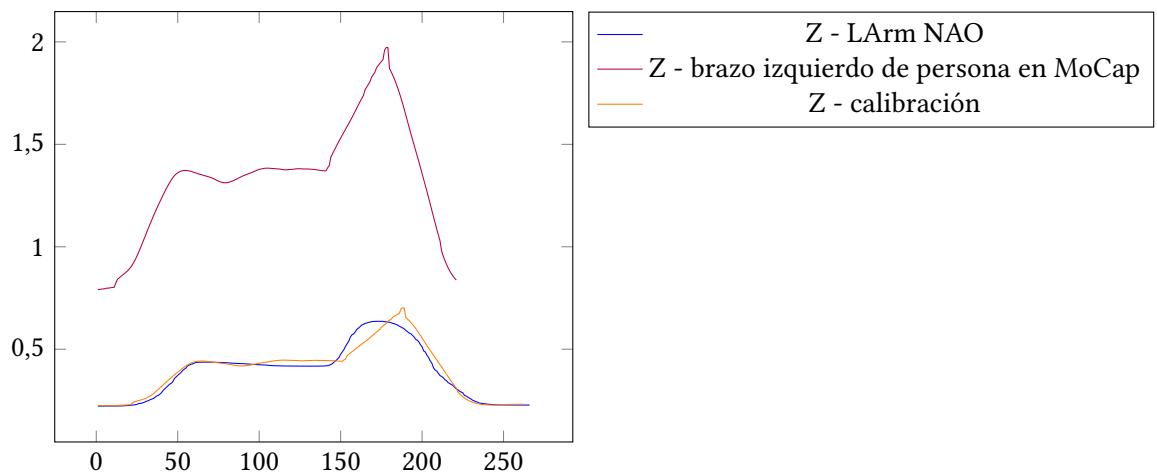


Figura A.51: Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

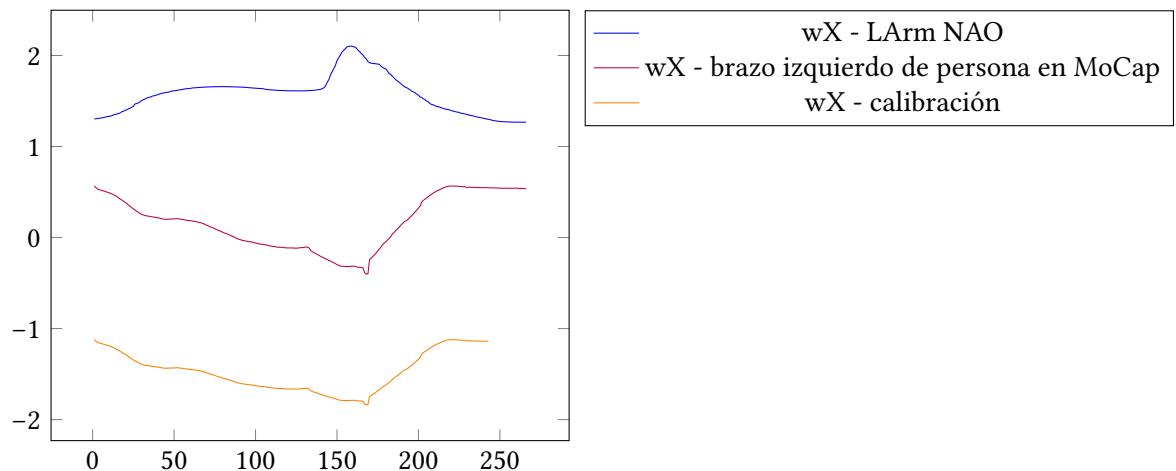


Figura A.52: Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

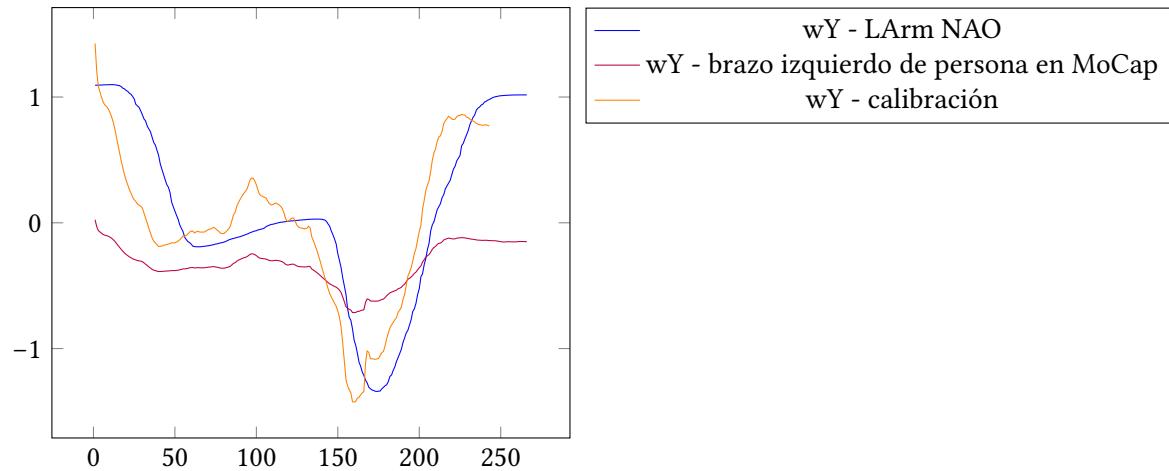


Figura A.53: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

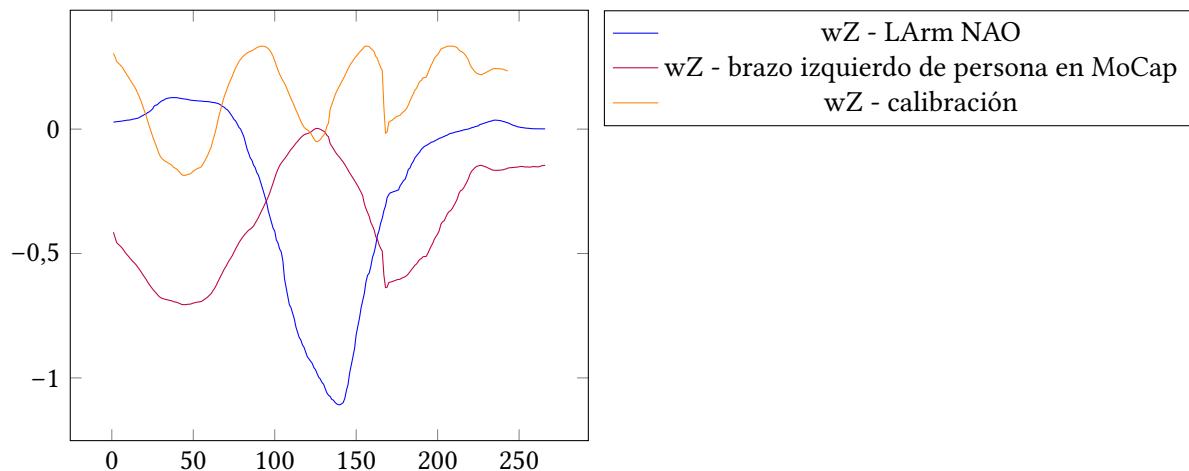


Figura A.54: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores LArm del robot humanoide NAO (brazo izquierdo). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

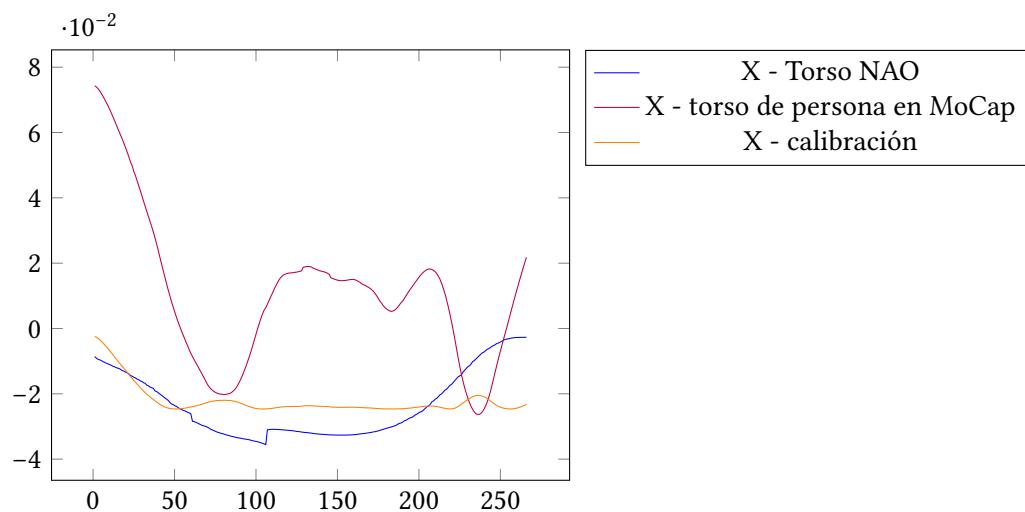
Torso

Figura A.55: Ajuste polinomio grado 2 en datos del eje X para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

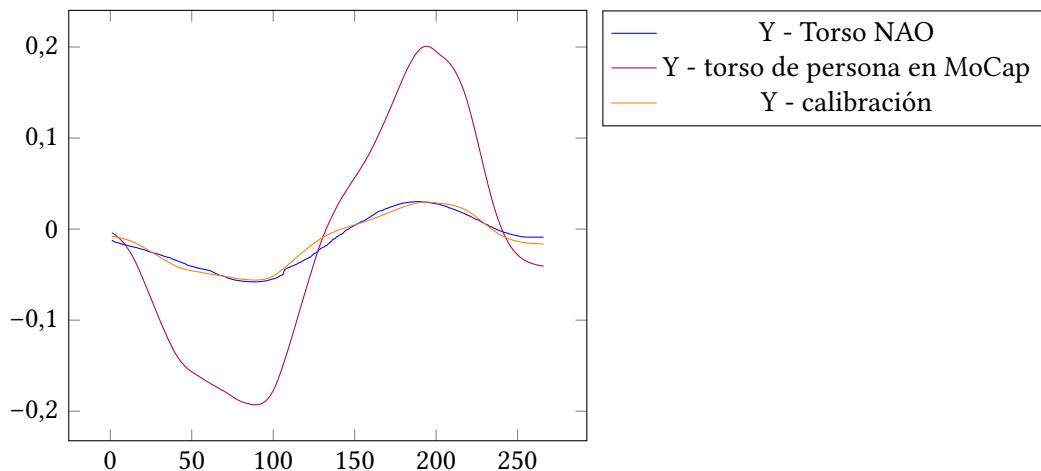


Figura A.56: Ajuste polinomio grado 2 en datos del eje Y para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

A. Resultados completos de la validación del sistema de teleoperación

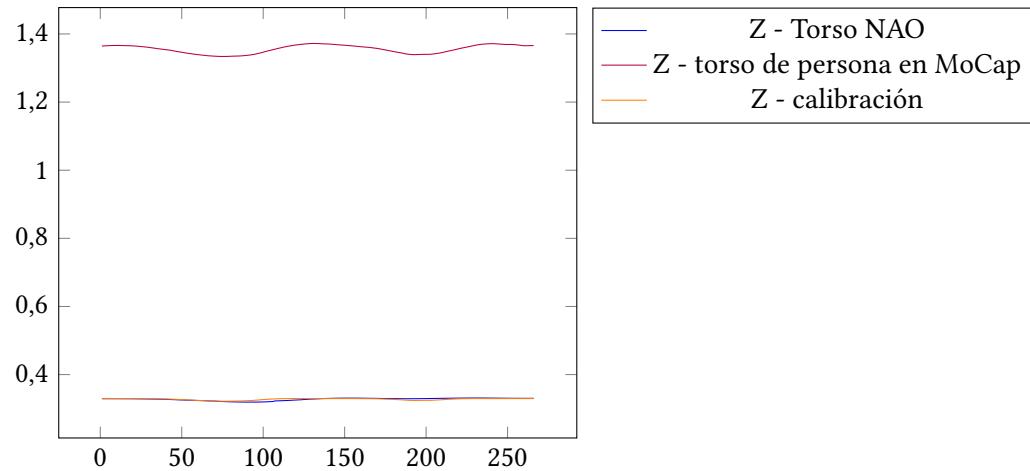


Figura A.57: Ajuste polinomio grado 2 en datos del eje Z para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

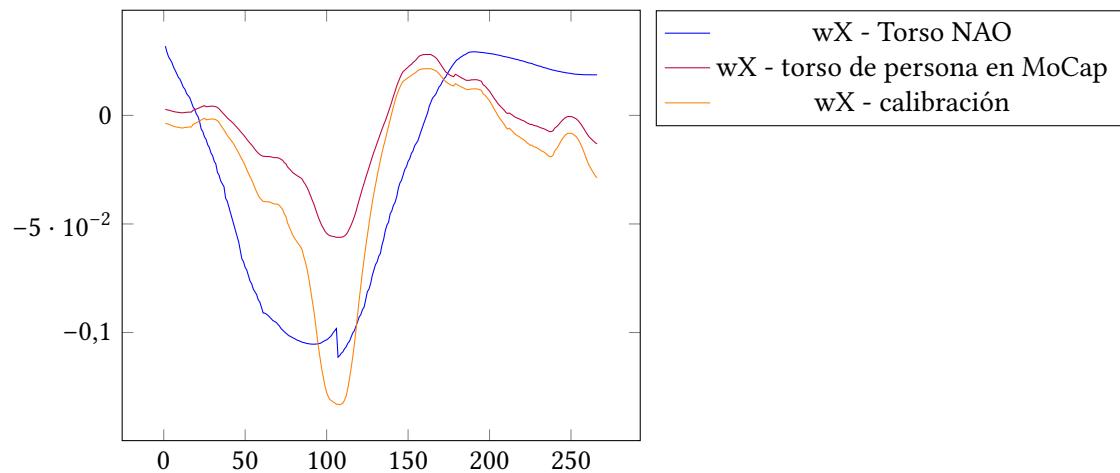


Figura A.58: Ajuste polinomio grado 2 en datos de rotación alrededor del eje X para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

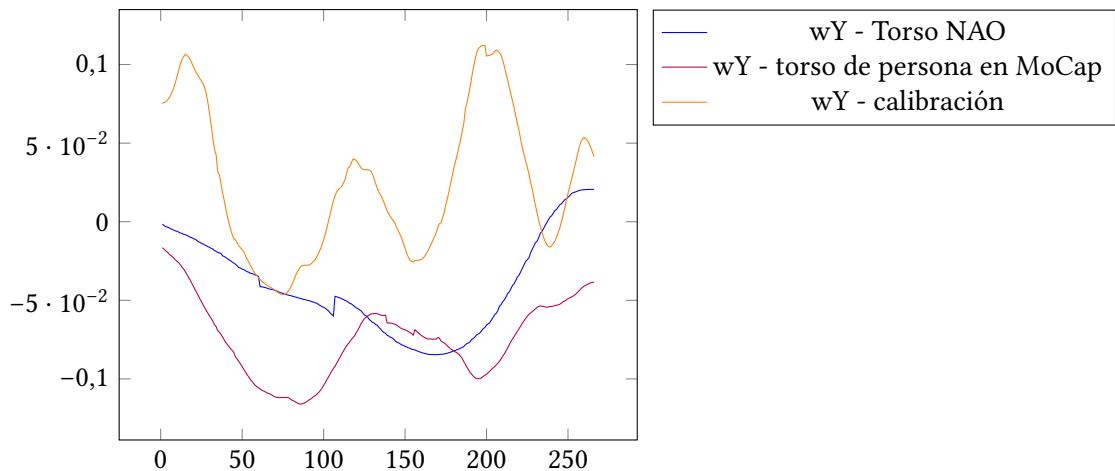


Figura A.59: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Y para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

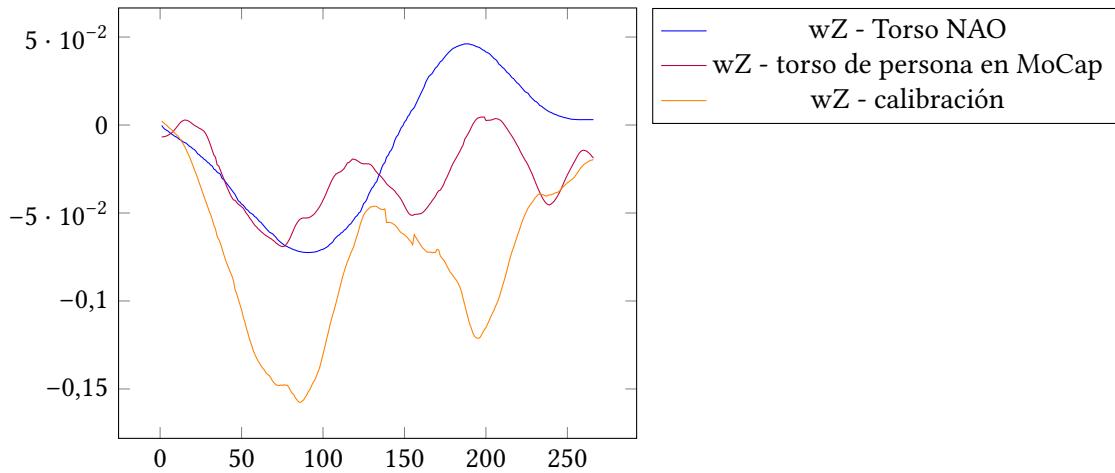


Figura A.60: Ajuste polinomio grado 2 en datos de rotación alrededor del eje Z para la cadena de actuadores Torso del robot humanoide NAO (torso). Marco de referencia ROBOT. Modelo de 6 cuerpos rígidos.

Apéndice B

Proceso de Calibración

Las rutinas que conforman la calibración iniciaron como parte del cumplimiento del plan de pruebas especificado en los objetivos del proyecto, se vio en estos el uso como parte de la calibración y por medio de pequeños ajustes en su ejecución se desarrolló el proceso presente.

Recordar que para el proceso de calibración se debe primero extraer los datos del NAO ejecutando las rutinas (estos datos ya son existentes en .../Code/Ver_Release/Calibration/NAO/RigidBody_Default, y no debería ser necesario generarlos nuevamente). Luego, la persona deberá ejecutar las mismas rutinas guiado por el robot NAO, es decir, la persona debe imitar al NAO para que concuerden los datos en la línea de tiempo, estos datos se deben exportar a CSV y guardar en .../Code/Ver_Release/Calibration/Human.

La calibración se divide en cuatro rutinas, una por sección a calibrar, cuyo objetivo es generar información de cambio de ubicación espacial para cada cadena actuadores a controlar. Para ejecutar una rutina se debe abrir desde el programa *Choregraphe*, el bloque de la animación debe estar conectado al anclaje de inicio y finalización. Si se quiere ejecutar la rutina más lento se debe editar la linea temporal y cambiar lo cuadros por segundo de la ejecución (FPS). Las rutinas están ubicadas en .../Code/Ver_Release/Calibration/Routines/.

A continuación se muestran imágenes con las posiciones clave de cada rutina ejecutada por el NAO, y su equivalente ejecutado por una persona. Todas las rutinas de calibración comienzan y terminan con la pose **Stand**.



Figura B.1: Pose Stand, inicial y final para todas las rutinas de calibración

B.1. Calibración de los Brazos (RArm y LArm)

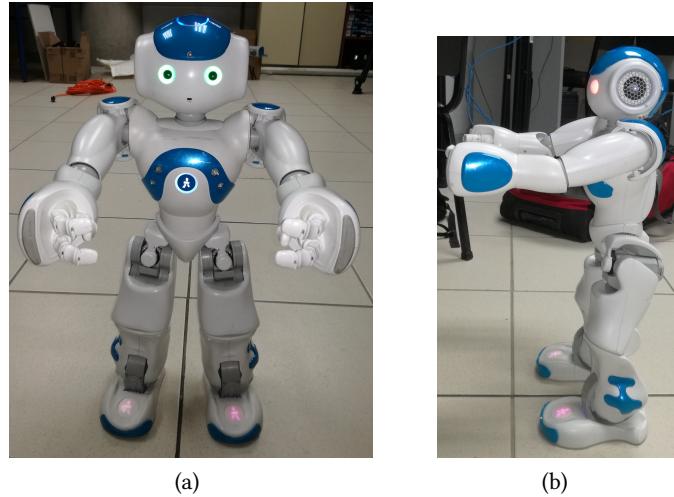


Figura B.2: Calibración de brazos. Paso 1, NAO.

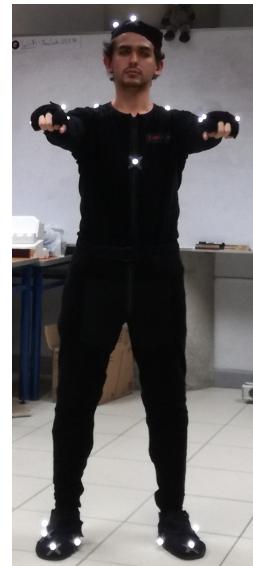
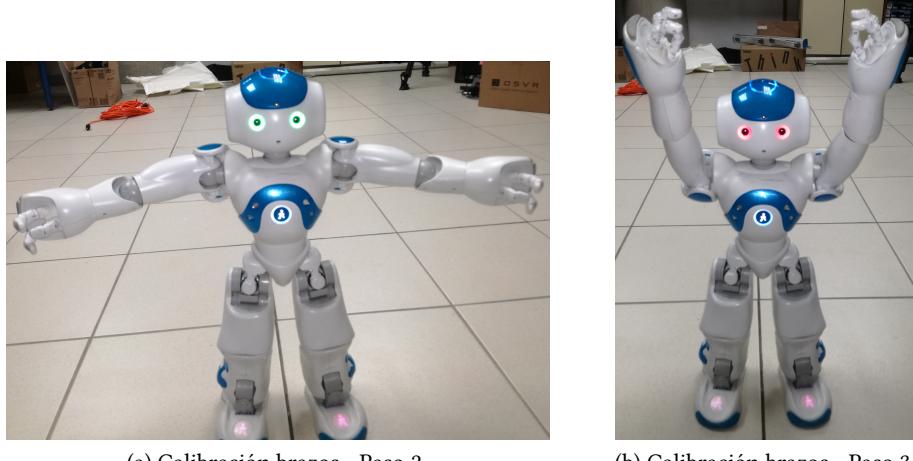


Figura B.3: Calibración de brazos. Paso 1, persona.



(a) Calibración brazos - Paso 2

(b) Calibración brazos - Paso 3

Figura B.4: Calibración de brazos. Pasos 2 y 3, NAO.



(a) Calibración brazos - Paso 2

(b) Calibración brazos -
Paso 3

Figura B.5: Calibración de brazos. Pasos 2 y 3, persona.

B.2. Torso



(a) Calibración torso - Paso 1

(b) Calibración torso - Paso 2

Figura B.6: Calibración del Torso. Pasos 1 y 2, NAO.



(a) Calibración torso - Paso
1

(b) Calibración torso - Paso
2

Figura B.7: Calibración del Torso. Pasos 1 y 2, persona.

B.3. Pierna Izquierda



(a) Calibración pierna izquierda - Paso 1



(b) Calibración pierna izquierda - Paso 2

Figura B.8: Pierna Izquierda. Pasos 1 y 2, NAO.



(a) Calibración pierna izquierda - Paso 1



(b) Calibración pierna izquierda - Paso 2

Figura B.9: Pierna Izquierda. Pasos 1 y 2, persona.

B.4. Pierna Derecha



Figura B.10: Pierna Derecha. Pasos 1 y 2, NAO.



Figura B.11: Pierna Derecha. Pasos 1 y 2, persona.

Apéndice C

Código del sistema de teleoperación

El código completo con demás archivos auxiliares como archivos CSV de comparaciones y gráficas se encuentran en el [repositorio git del proyecto](#), con la versión más reciente. Este apéndice incluye la versión del sistema al momento de creación de este documento.

C.1. ErrorFunc.py

Incluye un pequeño administrador para abortar los procesos ante errores de uso conocidos. Es una versión muy simple que en definitiva se puede mejorar, o bien eliminar su uso.

```
1 #####+
2 #Manejo de errores de uso de la aplicacion. Recibe explicacion del error,
3 #script donde se dio el error, y entrada recibida que ocasiono el error (opcional)
4 #####
5
6 #Imports
7 import sys
8
9 -----
10 -----
11 #abort: ocasiona un aborto de la operacion en proceso
12 def abort(explanation, value=None, program=None, called=None):
13     print "-----"
14     print "ERROR on Teleoperation Execution:"
15     if value is not None:
16         print "    Received", value
17         print "    ", explanation
18         if called is not None:
19             print "    Called from:", called
20         if program is not None:
21             print "    A borting program:", program
22         else:
23             print "    Aborting process"
24     else:
```

```
25     print "    ", explanation
26     if called is not None:
27         print "        Called from:", called
28     if program is not None:
29         print "        Aborting program:", program
30     else:
31         print "        Aborting process"
32     print "-----"
33     sys.exit()
34 #-----
```

C.2. Calibración

C.2.1. Calibrate.py

```

1 #####+
2 # Utiliza las funciones definidas en CalibrateFunc.py para realizar el proceso de
3 # calibracion para la teleoperacion del NAO, los factores generados por este
4 # proceso son usados en CSV_read.py para el ajuste de las coordenadas.
5 ##
6 # Realiza la calibracion segun las grabaciones de datos para calibrar dentro del
7 # directorio local ../Calibration/NAO para las del robot NAO, y ../Calibration/↔
8 # Human para las de
9 #la persona. Las grabaciones del NAO son predefinidas, las de la persona en
10 #utilizar el sistema deben obtenerse de grabaciones previas al proceso de
10 #calibracion.
11 ##
12 #El directorio donde se encuentran los datos de la persona se debe especificar
13 #y estar contenido en .../Calibration/Human con las grabaciones requeridas, cuyos ↔
13 #nombres
14 #deben seguir el formato *LetraMayusculaIndicandoPrueba*_Cal_P.csv, con la
15 #letra en mayuscula correspondiente a la pose de calibracion del NAO.
16 #Si no se especifica directorio se considera que se usan los archivos predefinidos
17 #dentro de .../Calibration/Human
18 #####+
19
20 #Imports
21 import sys
22 import time
23
24 ##Custom
25 import Calibrate_Func as cal
26 import Error_Func as error
27 import OffsetFile_Func as offset
28
29 -----
30 -----
31 def main(polDeg, wRot, humanDir, ID, naoDir):
32     #Datos para calibrar el movimiento de los brazos
33     ###Obtiene datos desde los csv correspondientes para la primer pose de ↔
34     #calibracion
34     RArmNaoA,RLegNaoA,LLegNaoA,LArmNaoA,TorsoNaoA,HeadNaoA,RArmPA,RLegPA,LLegPA,↔
34     LArmPA,TorsoPA,HeadPA = cal.setCalData( naoDir + "/" + "Brazos_NAO.csv", ↔
34     humanDir + "/" + "Brazos_P.csv", wRot)
35     ##Obtiene factores para la regresion polinomial deseada para el ajuste
36     factRArm = cal.getTerms(RArmNaoA, RArmPA, polDeg, wRot)
37     factLArm = cal.getTerms(LArmNaoA, LArmPA, polDeg, wRot)
38
39
40

```

```

41     ## Repite pasos anteriores para cada una de las poses usadas para los demás ↵
42     #actuadores
43     ### Pierna derecha
44     RArmNaoB, RLegNaoB, LLegNaoB, LArmNaoB, TorsoNaoB, HeadNaoB, RArmPB, RLegPB, LLegPB, ↵
45     LArmPB, TorsoPB, HeadPB = cal.setCalData( naoDir + "/" + "PiernaD_NAO.csv", ↵
46     humanDir + "/" + "PiernaD_P.csv", wRot)
47     #print RLegPB
48     factRLeg = cal.getTerms(RLegNaoB, RLegPB, polDeg, wRot)
49     ### Pierna izquierda
50     RArmNaoC, RLegNaoC, LLegNaoC, LArmNaoC, TorsoNaoC, HeadNaoC, RArmPC, RLegPC, LLegPC, ↵
51     LArmPC, TorsoPC, HeadPC = cal.setCalData( naoDir + "/" + "PiernaI_NAO.csv", ↵
52     humanDir + "/" + "PiernaI_P.csv", wRot)
53     factLLeg = cal.getTerms(LLegNaoB, LLegPB, polDeg, wRot)
54     ### Pierna Torso
55     RArmNaoD, RLegNaoD, LLegNaoD, LArmNaoD, TorsoNaoD, HeadNaoD, RArmPD, RLegPD, LLegPD, ↵
56     LArmPD, TorsoPD, HeadPD = cal.setCalData( naoDir + "/" + "Torso_NAO.csv", ↵
57     humanDir + "/" + "Torso_P.csv", wRot)
58     factTorso = cal.getTerms(TorsoNaoB, TorsoPB, polDeg, wRot)

59 #-----
60 #-----
61 #Genera archivo CSV con los offsets finales a utilizar
62 print "++++++"
63 print "Writing offsets to file: .../Calibration/Offsets/offsets_"+ID+".csv"
64 try:
65     offset.writeOffsets(polDeg, [factRArm, factRLeg, factLLeg, factLArm, ↵
66     factTorso], wRot, ID)
67 except Exception,e:
68     error.abort("Offset write unsuccessfull", "not valid parameters to ↵
69     function", "OffsetFileFunc", "Calibrate")
70 print "Done"
71 print "++++++"
72 print "++++++"
73 #-----
74 if __name__ == "__main__":
75     polDeg = 2 #Polinomio grado 2 por defecto
76     humanDir = ""
77     naoDir = "RigidBody_Default"
78     wRot = "yes"
79     ID = ""
80
81     if len(sys.argv) == 4:
82         try:
83             polDeg = int(sys.argv[1])
84             wRot = (sys.argv[2]).lower()
85             humanDir = sys.argv[3]
86             print "++++++"
87             print "Using Pol degree:", polDeg
88             print "Include rotations:", wRot

```

```

83     print "Reading Human data from dir:", humanDir
84     print "Reading NAO data from default directory .../Calibration/Nao/←
85         RigidBody_Default"
86     print "-----"
87     print "Starting calibration process..."
88     print "++++++"
89     except ValueError as e:
90         error.abort("Expected int as argument in main function", None, "←
91             Calibrate")
92     elif len(sys.argv) == 5:
93         try:
94             polDeg = int(sys.argv[1])
95             wRot = (sys.argv[2]).lower()
96             humanDir = sys.argv[3]
97             ID = sys.argv[4]
98             print "++++++"
99             print "Using Pol degree:", polDeg
100            print "Include rotations:", wRot
101            print "Reading Human data from dir:", humanDir
102            print "Reading NAO data from dir:", naoDir
103            print "Adding extension name to offsets file:", ID
104            print "-----"
105            print "Starting calibration process..."
106            print "++++++"
107            time.sleep(1.5) #Tiempo para que el usuario lea las indicaciones
108        except ValueError as e:
109            error.abort("Expected int as argument in main function", None, "←
110                Calibrate")
111    elif len(sys.argv) == 6:
112        try:
113            polDeg = int(sys.argv[1])
114            wRot = (sys.argv[2]).lower()
115            humanDir = sys.argv[3]
116            ID = sys.argv[4]
117            naoDir = sys.argv[5]
118            print "++++++"
119            print "Using Pol degree:", polDeg
120            print "Include rotations:", wRot
121            print "Reading Human data from dir:", humanDir
122            print "Reading NAO data from dir:", naoDir
123            print "Adding extension name to offsets file:", ID
124            print "-----"
125            print "Starting calibration process..."
126            print "++++++"
127            time.sleep(1.5) #Tiempo para que el usuario lea las indicaciones
128        except ValueError as e:
129            error.abort("Expected int as argument in main function", None, "←
130                Calibrate")
131    else:
132        error.abort("Expected 3, 4 or 5 arguments on call.", None, "Calibrate")
133

```

```
130     time.sleep(1.0)
131     main(polDeg, wRot, humanDir, ID, naoDir)
```

C.2.2. Calibratefunc.py

```
1 #####+
2 #Incluye las funciones a utilizar para el proceso de calibracion definido en
3 #Calibrate.py
4 #####
5
6 #Imports
7 import csv
8 import os
9 import numpy as np
10 from scipy.interpolate import *
11 from itertools import islice
12 from os.path import dirname, abspath
13 from copy import deepcopy
14
15 #Custom
16 import ErrorFunc as error
17 #-----
18 #-----
19 #Globales
20 ###Listas de coordenadas para cada actuador
21 RArmPCal = list()
22 RLegPCal = list()
23 LLegPCal = list()
24 LArmPCal = list()
25 TorsoPCal = list()
26 HeadPCal = list()
27 ###Listas de coordenadas para cada actuador
28 RArmNaoCal = list()
29 RLegNaoCal = list()
30 LLegNaoCal = list()
31 LArmNaoCal = list()
32 TorsoNaoCal = list()
33 HeadNaoCal = list()
34
35 #-----
36 #-----
37 #Recibe archivos con las grabaciones de calibracion de una pose, del NAO y de
38 #la persona. Devuelve listas individuales para cada actuador del NAO y de la
39 #persona.
40 def setCalData(archNao, archP, wRot):
41     #Limpia listas por usos previos
42     RArmPCal[:] = []
43     RLegPCal[:] = []
44     LLegPCal[:] = []
45     LArmPCal[:] = []
46     TorsoPCal[:] = []
47     HeadPCal[:] = []
48     RArmNaoCal[:] = []
```

```

49     RLegNaoCal[:] = []
50     LLegNaoCal[:] = []
51     LArmNaoCal[:] = []
52     TorsoNaoCal[:] = []
53     HeadNaoCal[:] = []
54
55     # Directorios con los datos de calibracion
56     ##Root (./ Code)
57     rootDir = dirname(dirname(abspath(__file__)))
58     ##Archivos con posiciones del Nao
59     dirNao = os.path.join(rootDir, "Ver5/Cal/NAO/")
60     dirNao = os.path.join(dirNao, archNao)
61     ##Archivos con posiciones de la PERSONA_ROBOT
62     dirPersona = os.path.join(rootDir, "Ver5/Cal/Human/")
63     dirPersona = os.path.join(dirPersona, archP)
64
65     #Obteniendo contenidos
66     ##Nao
67     try:
68         fNao = open(dirNao, 'rt')
69     except Exception,e:
70         error.abort("is not a valid directory", archNao, "Calibrate")
71
72     ##Obteniendo datos completos y cerrando archivo
73     reader = csv.reader(fNao)
74     filasNao = [r for r in reader]
75     fNao.close()
76     ##Persona
77     try:
78         fPersona = open(dirPersona, 'rt')
79     except Exception,e:
80         error.abort("is not a valid directory", archP, "Calibrate")
81     ##Obteniendo datos completos y cerrando archivo
82     reader = csv.reader(fPersona)
83     filasPersona = [r for r in reader]
84     fPersona.close()
85
86     #↔
87     #↔
88
89     #Extraccion datos del Nao
90     ##Datos numericos a partir de la fila #8
91     filasDatosNao = filasNao[7::]
92     ##Eliminando columnas de Cuadro y Tiempos
93     for i,item in enumerate(filasDatosNao):
94         del filasDatosNao[i][0]
95         del filasDatosNao[i][0]

```

```

96     if wRot.lower() == "no":
97         #Sin rotaciones
98         contXYZ = 0
99         trioXYZ = [0.0 ,0.0 ,0.0]
100        contAct = 0
101
102        for i,item in enumerate(filasDatosNao):
103            for contTrio in range(0,18): #3DoF*6 Actuadores
104                if contXYZ < 2:
105                    try:
106                        trioXYZ[contXYZ] = float(filasDatosNao[i].pop(0))
107                    except ValueError:
108                        print "Cell ignored"
109                        contXYZ+=1
110                elif contXYZ == 2:
111                    try:
112                        trioXYZ[contXYZ] = float(filasDatosNao[i].pop(0))
113                    except ValueError:
114                        print "Cell ignored"
115                    contXYZ = 0
116
117                if (contAct == 0):
118                    RArmNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2]])
119                elif (contAct == 1):
120                    RLegNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2]])
121                elif (contAct == 2):
122                    LLegNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2]])
123                elif (contAct == 3):
124                    LArmNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2]])
125                elif (contAct == 4):
126                    TorsoNaoCal.append([trioXYZ[0], trioXYZ[1], trioXYZ[2]])
127                elif (contAct == 5):
128                    HeadNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2]])
129
130                if (contAct == 5):
131                    contAct = 0
132                else:
133                    contAct+=1
134
135        elif wRot.lower() == "yes":
136            #Con rotaciones
137            contXYZ = 0
138            trioXYZ = [0.0 ,0.0 ,0.0 ,0.0 ,0.0 ,0.0 ]
139            contAct = 0
140
141            for i,item in enumerate(filasDatosNao):
142                for contTrio in range(0,36):#6Dof*6 Actuadores
143                    if contXYZ < 5:
144                        try:
145                            trioXYZ[contXYZ] = float(filasDatosNao[i].pop(0))
146                        except ValueError:

```

```

147         print "Cell ignored"
148
149         contXYZ+=1
150     elif contXYZ == 5:
151         try:
152             trioXYZ[contXYZ] = float(filasDatosNao[i].pop(0))
153         except ValueError:
154             print "Cell ignored"
155
156         contXYZ = 0
157
158     if (contAct == 0):
159         RArmNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2], ←
160                             trioXYZ[3], trioXYZ[4], trioXYZ[5]])
161     elif (contAct == 1):
162         RLegNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2], ←
163                             trioXYZ[3], trioXYZ[4], trioXYZ[5]])
164     elif (contAct == 2):
165         LLegNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2], ←
166                             trioXYZ[3], trioXYZ[4], trioXYZ[5]])
167     elif (contAct == 3):
168         LArmNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2], ←
169                             trioXYZ[3], trioXYZ[4], trioXYZ[5]])
170     elif (contAct == 4):
171         TorsoNaoCal.append([trioXYZ[0], trioXYZ[1], trioXYZ[2], ←
172                             trioXYZ[3], trioXYZ[4], trioXYZ[5]])
173     elif (contAct == 5):
174         HeadNaoCal.append( [trioXYZ[0], trioXYZ[1], trioXYZ[2], ←
175                             trioXYZ[3], trioXYZ[4], trioXYZ[5]])
176
177     if (contAct == 5):
178         contAct = 0
179     else:
180         contAct+=1
181
182 #←
183 -----←
184
185 #Extraccion datos de la persona
186 #NOTA: ejes Y y Z estan invertidos
187 ##Obencion del orden de aparicion de los marcadores(actuadores)
188 filasActuadores = filasPersona[3]
189 ##Remueve primeros dos espacios siempre en blanco
190 filasActuadores.remove(' ')
191 filasActuadores.remove(' ')
192 j = 0
193 listaActuadores = [None]*6 #Se trabaja con cadenas de accion del NAO
194 if wRot.lower() == "no":
195     for i, item in enumerate(filasActuadores):
196         #Se repite el nombre del marcador 3 veces(XYZ)
197         if i==0 or i==3 or i==6 or i==9 or i==12 or i==15:

```

```

190         listaActuadores[j] = str(item)
191         j+=1
192     elif wRot.lower() == "yes":
193         for i, item in enumerate(filasActuadores):
194             #Se repite el nombre del marcador 3 veces(XYZ)
195             if i==0 or i==8 or i==16 or i==24 or i==32 or i==40:
196                 listaActuadores[j] = str(item)
197                 j+=1
198
199     ##Datos numericos a partir de la fila #8
200     filasDatosP = filasPersona[7::]
201     ##Eliminando columnas de Cuadro y Tiempos
202     for i,item in enumerate(filasDatosP):
203         del filasDatosP[i][0]
204         del filasDatosP[i][0]
205
206     #Sin rotaciones
207     if wRot.lower() == "no":
208         contXYZ = 0
209         trioXYZ = [0.0,0.0,0.0]
210         contAct = 0
211
212         for i,item in enumerate(filasDatosP):
213             for contTrio in range(0,18):
214                 if contXYZ < 2:
215                     try:
216                         trioXYZ[contXYZ] = float(filasDatosP[i].pop(0))
217                     except ValueError:
218                         print "Cell ignored"
219                     contXYZ+=1
220                 elif contXYZ == 2:
221                     try:
222                         trioXYZ[contXYZ] = float(filasDatosP[i].pop(0))
223                     except ValueError:
224                         print "Cell ignored"
225                     contXYZ = 0
226
227                 if listaActuadores[contAct] == "RArm":
228                     RArmPCal.append([-1*trioXYZ[0], trioXYZ[2], trioXYZ[1]])
229                 elif listaActuadores[contAct] == "RLeg":
230                     RLegPCal.append([-1*trioXYZ[0], trioXYZ[2], trioXYZ[1]])
231                 elif listaActuadores[contAct] == "LLeg":
232                     LLegPCal.append([-1*trioXYZ[0], trioXYZ[2], trioXYZ[1]])
233                 elif listaActuadores[contAct] == "LArm":
234                     LArmPCal.append([-1*trioXYZ[0], trioXYZ[2], trioXYZ[1]])
235                 elif listaActuadores[contAct] == "Torso":
236                     TorsoPCal.append([-1*trioXYZ[0], trioXYZ[2], trioXYZ[1]])
237                 elif listaActuadores[contAct] == "Head":
238                     HeadPCal.append([-1*trioXYZ[0], trioXYZ[2], trioXYZ[1]])
239
240                 if (contAct == 5):

```

```

241             contAct = 0
242         else:
243             contAct+=1
244 #Con rotaciones
245 elif wRot.lower() == "yes":
246     contXYZ = 0
247     trioXYZ = [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0] #Incluye wX,wY,wZ,W,X,Y,Z,↔
248     Error
249     contAct = 0
250     temp = 0
251
252     for i,item in enumerate(filasDatosP):
253         for contTrio in range(0,48):
254             if contXYZ < 7:
255                 temp = filasDatosP[i].pop(0)
256                 try:
257                     trioXYZ[contXYZ] = float(temp)
258                 except ValueError:
259                     #Es una celda marcada como invalida, luego se procesa
260                     ##Las celdas invalidas no se usan para calibracion porque ↔
261                     #no dan
262                     ##un buen ajuste temporal entre los datos, se marcan con ↔
263                     #la palabra
264                     #'Empty' de manera manual en el CSV a leer
265                     trioXYZ[contXYZ] = temp
266             contXYZ+=1
267     elif contXYZ == 7:
268         temp = filasDatosP[i].pop(0)
269         try:
270             trioXYZ[contXYZ] = float(temp)
271         except ValueError:
272             trioXYZ[contXYZ] = temp
273         contXYZ = 0
274         #Y y Z estan intercambiadas, al igual que wY y wZ
275         #Solo se ocupan X,Y,Z,wX,wY,wZ
276         if listaActuadores[contAct] == "RArm":
277             RArmPCal.append([trioXYZ[4],trioXYZ[6],trioXYZ[5],trioXYZ↔
278                             [0],trioXYZ[2],trioXYZ[1]])
279         elif listaActuadores[contAct] == "RLeg":
280             RLegPCal.append([trioXYZ[4],trioXYZ[6],trioXYZ[5],trioXYZ↔
281                             [0],trioXYZ[2],trioXYZ[1]])
282         elif listaActuadores[contAct] == "LLeg":
283             LLegPCal.append([trioXYZ[4],trioXYZ[6],trioXYZ[5],trioXYZ↔
284                             [0],trioXYZ[2],trioXYZ[1]])
285         elif listaActuadores[contAct] == "LArm":
286             LArmPCal.append([trioXYZ[4],trioXYZ[6],trioXYZ[5],trioXYZ↔
287                             [0],trioXYZ[2],trioXYZ[1]])
288         elif listaActuadores[contAct] == "Torso":
289             TorsoPCal.append([trioXYZ[4],trioXYZ[6],trioXYZ[5],trioXYZ↔
290                             [0],trioXYZ[2],trioXYZ[1]])
291         elif listaActuadores[contAct] == "Head":
292

```

```

284             HeadPCal.append( [trioXYZ[4],trioXYZ[6],trioXYZ[5],trioXYZ<-
285                             [0], trioXYZ[2], trioXYZ[1]])  

286             if (contAct == 5):
287                 contAct = 0
288             else:
289                 contAct+=1  

290
291     return RArmNaoCal,RLegNaoCal,LLegNaoCal,LArmNaoCal,TorsoNaoCal,HeadNaoCal,<-
292           RArmPCal,RLegPCal,LLegPCal,LArmPCal,TorsoPCal,HeadPCal  

293 #-----  

294 #-----  

295 # Realiza la regresion polinomial deseada para encontrar los terminos del ajuste  

296 # Recibe la lista con datos de coordenadas XYZ para un solo actuador, del NAO  

297 #y de la persona, asi como el grado polinomial.  

298 #Devuelve una lista con los terminos del ajuste polinomial para X, Y y Z del  

299 #actuador recibido  

300 def getTerms(listNAO, listP, degree, wRot):  

301     naoX = list()  

302     naoY = list()  

303     naoZ = list()  

304     naowX = list()  

305     naowY = list()  

306     naowZ = list()  

307     pX = list()  

308     pY = list()  

309     pZ = list()  

310     pwX = list()  

311     pwY = list()  

312     pwZ = list()  

313     dif = 0  

314
315     if wRot.lower() == "no":
316         pl = [None]*3
317     elif wRot.lower() == "yes":
318         pl = [None]*6  

319
320 #-----  

321 #Separa datos X, Y y Z en grupos  

322
323     if wRot.lower() == "no":
324         #Datos NAO
325         for i in range(len(listNAO)):
326             naoX.append(listNAO[i][0])
327             naoY.append(listNAO[i][1])
328             naoZ.append(listNAO[i][2])
329         #Datos MoCap
330         for i in range(len(listP)):
331             pX.append(listP[i][0])
332             pY.append(listP[i][1])

```

```

333         pZ.append(listP[i][2])
334     elif wRot.lower() == "yes":
335         #Datos NAO
336         for i in range(len(listNAO)):
337             naoX.append(listNAO[i][0])
338             naoY.append(listNAO[i][1])
339             naoZ.append(listNAO[i][2])
340             naowX.append(listNAO[i][3])
341             naowY.append(listNAO[i][4])
342             naowZ.append(listNAO[i][5])
343             #Datos MoCap
344         for i in range(len(listP)):
345             ##X
346             if isinstance(listP[i][0], float):
347                 pX.append(listP[i][0])
348             ##Y
349             if isinstance(listP[i][1], float):
350                 pY.append(listP[i][1])
351             ##Z
352             if isinstance(listP[i][2], float):
353                 pZ.append(listP[i][2])
354             ##wX
355             if isinstance(listP[i][3], float):
356                 pwX.append(listP[i][3])
357             ##wY
358             if isinstance(listP[i][4], float):
359                 pwY.append(listP[i][4])
360             ##wZ
361             if isinstance(listP[i][5], float):
362                 pwZ.append(listP[i][5])
363
364 #Manejo de diferentes longitudes de las listas , hace que las listas tengan
365 #la misma longitud si correrlas temporalmente , segun como estaban ↪
366 #acomodadas
367 #desde la comparacion grafica
368 ###X
369 dif = len(pX)-len(naoX)
370 if dif < 0:
371     del naoX[len(naoX)-1+dif:len(naoX)-1]
372 elif dif > 0:
373     del pX[len(pX)-1-dif:len(pX)-1]
374 ###Y
375 dif = len(pY)-len(naoY)
376 if dif < 0:
377     del naoY[len(naoY)-1+dif:len(naoY)-1]
378 elif dif > 0:
379     del pY[len(pY)-1-dif:len(pY)-1]
380 ###Z
381 dif = len(pZ)-len(naoZ)
382 if dif < 0:
383     del naoZ[len(naoZ)-1+dif:len(naoZ)-1]

```

```

383     elif dif > 0:
384         del pZ[len(pZ)-1-dif:len(pZ)-1]
385     ##wX
386     dif = len(pwX)-len(naowX)
387     if dif < 0:
388         del naowX[len(naowX)-1+dif:len(naowX)-1]
389     elif dif > 0:
390         del pwX[len(pwX)-1-dif:len(pwX)-1]
391     ##wY
392     dif = len(pwY)-len(naowY)
393     if dif < 0:
394         del naowY[len(naowY)-1+dif:len(naowY)-1]
395     elif dif > 0:
396         del pwY[len(pwY)-1-dif:len(pwY)-1]
397     ##wZ
398     dif = len(pwZ)-len(naowZ)
399     if dif < 0:
400         del naowZ[len(naowZ)-1+dif:len(naowZ)-1]
401     elif dif > 0:
402         del pwZ[len(pwZ)-1-dif:len(pwZ)-1]
403
404 -----
405 #Obtiene los terminos de la relacion polinomial con grado 'degree'
406 ##Se ocupa que ambos arreglos a comparar tengan la misma cantidad de datos,
407 ##por lo que se supone que en el estudio de comparaciones se ajusto las
408 ##longitudes para que sean iguales
409
410 pl[0] = list(np.polyfit(px,naox,degree))
411 pl[1] = list(np.polyfit(py,naoy,degree))
412 pl[2] = list(np.polyfit(pz,naoz,degree))
413
414 if wRot.lower() == "yes":
415     pl[3] = list(np.polyfit(pwX,naowX,degree))
416     pl[4] = list(np.polyfit(pwY,naowY,degree))
417     pl[5] = list(np.polyfit(pwZ,naowZ,degree))
418
419 return pl

```

C.2.3. OffsetFileFunc.py

```

1 #####+
2 # Contiene funciones para manipular el archivo CSV con los offsets de la
3 # calibracion , se cumple el siguiente formato:
4 #####
5 #   Encabezado:      grado , TerminoN , TerminoN-1, ... , Termino1
6 #   Contenido:      Eje_Actuador ,   ValorN ,   ValorN-1, ... , Valor1
7 #####
8 #Con N = grado + 1
9 ##
10 #El archivo CSV con los offsets se sobreescribe cada vez que se hace una
11 #calibracion . Se guarda en el directorio .../Cal/Offsets , bajo el nombre
12 #no alterable " offsets .csv " .
13 #####
14
15 #Imports
16 import csv
17 import os
18 import time
19 from itertools import islice
20 from os.path import dirname, abspath
21
22 ##Custom
23 import ErrorFunc as error
24
25 #####
26 #####
27 #Lee archivo con los offsets y devuelve lista con el conjunto de terminos para
28 #cada actuador y el grado polinomial utilizado en la calibracion
29 def getOffsets():
30     print "Reading offsets values from default .../Cal/Offsets/offsets.csv"
31     #Root
32     archivo = dirname(dirname(abspath(__file__)))
33     archivo += "/Ver5/Cal/Offsets/offsets.csv"
34     ##Abriendo archivo
35     f = open(archivo, 'rt')
36     ##Obteniendo datos completos y cerrando archivo
37     reader = csv.reader(f)
38     filas = [r for r in reader]
39     f.close()
40     #####
41     ##Obtiene grado del polinomio
42     polDegree = int(filas[0].pop(0))
43     isRot = filas[0].pop(0)
44     ##Borrando encabezado
45     del filas[0]
46     #####
47     ##Cada elemento de "filas" incluye todos los terminos segun el grado ↔
        polinomial

```

```
48     ##por eje_actuador , segun el orden preferente (XYZ para ejes , RArm, RLeg ,
49     #LLeg , LArm, Torso , Head para actuadores; la lista es de 18 elementos)
50     print "Got offsets"
51     return filas,polDegree,isRot
52
53 #-----
54 #-----
55 #Recibe grado polinomial usado y lista con grupos de terminos por eje_actuador ,
56 #segun el orden preferente (XYZ para ejes , RArm, RLeg,#LLeg , LArm, Torso , Head
57 #para actuadores; la lista debe contener 18 elementos)
58 def writeOffsets(degree, eje_actuador, rotacion):
59     archivo = dirname(dirname(abspath(__file__)))
60     archivo += "/Ver5/Cal/Offsets/offsets.csv"
61     with open(archivo, 'w') as csvfile:
62         writer = csv.writer(csvfile)
63         #Encabezado (solo importa el grado ya que el orden de los terminos se
64         #considera siempre estaran bien)
65         writer.writerow([degree, rotacion])
66         for item in eje_actuador:
67             for element in item:
68                 writer.writerow(element)
69     return 0
```

C.3. Obtención datos del NAO

C.3.1. GetPositions.py

```

1 #+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2 #Al ejecutarse inicia la toma de datos de las posiciones de los actuadores
3 #correspondientes a los 6 efectores finales del robot humanoide NAO
4 #(RArm, RLeg, LLeg, LArm, Torso, Head). Se toman "rows" cantidad de sets de
5 #coordenadas. Se incluye coordenadas XYZ (+rotaciones si se especifica por el
6 #usuario)
7 #+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
8
9 #Imports
10 import sys
11 import time
12 from naoqi import ALProxy
13 import motion
14 import csv
15 import os
16 from itertools import islice
17 from os.path import dirname, abspath
18
19 ##Custom
20 import Error_Func as error
21
22 #-----
23 #-----
24 #Recibe IP del robot NAO, marco de referencia de las coordenadas y si se quiere
25 #tomar datos de las rotaciones.
26 #Devuelve un archivo CSV con la informacion solicitada , sigueiente el formato
27 #general de exportacion de archivo CSV de Motive
28 def main(robotIP, frameRef, rotation, name=None):
29     PORT = 9559
30     print "Using Port:", PORT
31     if rotation.lower() != "no":
32         if rotation.lower() != "yes":
33             error.abort("Expected a yes or no input for rotation data inclusion",  

34                         "GetPositions")
35
36     try:
37         print "Trying to create ALMotion proxy"
38         motionProxy = ALProxy("ALMotion", robotIP, PORT)
39     except Exception,e:
40         print "Could not create proxy to ALMotion"
41         print "Error was: ", e
42         sys.exit(1)
43
44 #Ajustes iniciales

```

```

45
46     #Marco de referencia para la obtencion de datos
47     if frameRef.upper() == "ROBOT":
48         space = motion.FRAME_ROBOT
49     elif frameRef.upper() == "TORSO":
50         space = motion.FRAME_TORSO
51     else:
52         error.abort("is not a valid frame for function", frame, "GetPositions")
53     #Uso de sensores adicionales para aproximar el estado del actuador
54     useSensorValues = False
55 #-----
56 #Obtencion de los datos
57
58     posRArm = []
59     posRLeg = []
60     posLLeg = []
61     posLArm = []
62     posTorso = []
63     posHead = []
64     rows = 0
65
66     print "++++++++++++++++++++++++++++++++++++++"
67     print "Starting to collect data, interrupt the process only if necessary"
68     print "Wait for completion"
69     print "Press Ctrl+C to stop collecting data and proceed with data writting"
70     time.sleep(0.5)
71
72     try:
73         while (True):
74             posRArm.append(motionProxy.getPosition("RArm", space, useSensorValues)↔
75                           )
75             posRLeg.append(motionProxy.getPosition("RLeg", space, useSensorValues)↔
76                           )
76             posLLeg.append(motionProxy.getPosition("LLeg", space, useSensorValues)↔
77                           )
77             posLArm.append(motionProxy.getPosition("LArm", space, useSensorValues)↔
78                           )
78             posTorso.append(motionProxy.getPosition("Torso", space, ↵
79                             useSensorValues))
79             posHead.append(motionProxy.getPosition("Head", space, useSensorValues)↔
80                           )
80             time.sleep(0.003)
81
82             rows +=1
83     except KeyboardInterrupt:
84         print "Data collection finished"
85         print "++++++++++++++++++++++++++++++++++++++"
86         print "Writing CSV with data"
87         time.sleep(0.25)
88         pass
89 #-----

```



```

123             'X Torso': posTorso[i][0], 'Y Torso': posTorso[←
124             i][1], 'Z Torso': posTorso[i][2], 'WX ←
125             Torso': posTorso[i][3], 'WY Torso': ←
126             posTorso[i][4], 'WZ Torso': posTorso[i←
127             ][5],
128             'X Head': posHead[i][0], 'Y Head': posHead[i←
129             ][1], 'Z Head': posHead[i][2], 'WX Head': ←
130             posHead[i][3], 'WY Head': posHead[i][4], '←
131             WZ Head': posHead[i][5],
132         })
133     #Archivo sin rotaciones
134 else:
135     with open(archivo, 'w') as csvfile:
136         fieldnames = ['C1', 'C2', 'X RArm', 'Y RArm', 'X RLeg', ←
137             'Y RLeg', 'Z RArm', 'X LLeg', 'Y LLeg', 'Z LLeg', 'X LArm', 'Y←
138             LArm', 'Z LArm',
139             'X Torso', 'Y Torso', 'Z Torso', 'X Head', 'Y ←
140             Head', 'Z Head']
141         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
142         for x in range(6):
143             writer.writerow({})
144         writer.writeheader()
145         for i in range(rows):
146             writer.writerow({'X RArm': posRArm[i][0], 'Y RArm': posRArm[←
147             i][1], 'Z RArm': posRArm[i][2], 'X RLeg': posRLeg[i][0], 'Y←
148             RLeg': posRLeg[i][1], 'Z RLeg': posRLeg[i][2],
149             'X LLeg': posLLeg[i][0], 'Y LLeg': posLLeg[←
150             i][1], 'Z LLeg': posLLeg[i][2], 'X ←
151             LArm': posLArm[i][0], 'Y LArm': ←
152             posLArm[i][1], 'Z LArm': posLArm[i←
153             ][2],
154             'X Torso': posTorso[i][0], 'Y Torso': ←
155             posTorso[i][1], 'Z Torso': posTorso[←
156             i][2], 'X Head': posHead[i][0], 'Y Head': ←
157             posHead[i][1], 'Z Head': posHead[i←
158             ][2],
159         })
160
161     print "END"
162     print "++++++"
163 #-----
164 #
165 if __name__ == "__main__":
166     robotIP = "10.0.1.128" #Bato en red PrisNAO
167     frameRef = "ROBOT"
168     rotation = "no"
169     name = None
170
171     if len(sys.argv) == 4:
172         robotIP = sys.argv[1]
173         frame = sys.argv[2]

```

```
154     rotation = sys.argv[3]
155     print "++++++"
156     print "Using robot IP:", sys.argv[1], " with frame:", frameRef
157     print "Include rotations:", rotation
158     print "Name of CSV file default by system date and time"
159     print "-----"
160     print "Starting to retrieve sensor values"
161     print "++++++"
162 elif len(sys.argv) == 5:
163     robotIP = sys.argv[1]
164     frame = sys.argv[2]
165     rotation = sys.argv[3]
166     name = sys.argv[4]
167     print "++++++"
168     print "Using robot IP:", sys.argv[1], " with frame:", frameRef
169     print "Include rotations:", rotation
170     print "Name of CSV file: NAO_", name
171     print "-----"
172     print "Starting to retrieve sensor values"
173     print "++++++"
174 else:
175     error.abort("Expected 3 or 4 arguments on call.", "GetPositions")
176
177 time.sleep(1.0)
178 main(robotIP, frameRef, rotation, name)
```

C.4. Ejecución de la teleoperación

C.4.1. Move.py

```
1 #####+
2 #Programa principal para la ejecucion de la Teleoperacion del robot NAO.
3 #Se encarga del control del NAO por grabaciones guardadas.. Recibe vectores
4 #de tiempos y coordenadas listos para ser enviados directamente al robot NAO.
5 #Solicita al usuario IP del robot a conectarse y nombre del archivo CSV que
6 #contiene los datos de la grabacion de MoCap que se quiere utilizar para que el
7 #robot "imita" los movimientos. (Primer argumento es el nombre del archivo y
8 #el segundo es la direccion IP)
9 ##
10 # Utiliza los API's del robot NAO para efectuar su control.
11 ##
12 #Permite utilizar marco de referencia ROBOT y TORSO, segun defina el usuario , por
13 #defecto se trabaja con ROBOT.
14 ##
15 #Hace uso del balanceador de cuerpo completo incluido en las herramientas de
16 #desarrollo de las librerias del robot humanoide NAO. Para realizar un mejor
17 #balance se requiere manipular los puntos de apoyo del balance segun la
18 #posicion que se esta llevando a cabo.
19 #####
20 # -*- encoding: UTF-8 -*-
21 #Imports
22 import sys
23 import time
24 import numpy as np
25 from naoqi import ALProxy
26 import motion
27
28 ##Custom
29 import CSVMOCAP_Func as csvMocap
30 import Error_Func as error
31
32 -----
33 -----
34 # Control de la rigidez de los motores
35 def StiffnessOn(proxy):
36     # Body representa elconjunto de todas las articulaciones
37     pNames = "Body"
38     pStiffnessLists = 1.0
39     pTimeLists = 1.0
40     proxy.stiffnessInterpolation(pNames, pStiffnessLists, pTimeLists)
41 -----
42 def main(robotIP,coreo,marcoRef,offFile):
43     #SetUp
44     PORT = 9559 #Puerto por defecto
45     print "+++++++"
```

```

46     print "Creating NAO proxies for posture and movement"
47 #creacion de objetos para usar los metodos de los API's del Nao
48 try:
49     #Para poder utilizar funciones de movimiento de los actuadores del NAO
50     motionProxy = ALProxy("ALMotion", robotIP, PORT)
51 except Exception,e:
52     error.abort("Could not create proxy to ALMotion.", None, "Move")
53 try:
54     #Para utilizar posiciones preestablecidas del NAO
55     postureProxy = ALProxy("ALRobotPosture", robotIP, PORT)
56 except Exception, e:
57     error.abort("Could not create proxy to ALRobotPosture.", None, "Move")
58 print "Connection to NAO available."
59 print "+++++++"+
60 #-----
61 #-----
62 #Inicializacion de parametros necesarios para el movimiento del NAO
63     print "-----"
64     print "      Setting frame"
65     ##Marco de Referencia a utilizar
66     referencia = motion.FRAME_TORSO #Referencia centro del Torsó
67                         #Vale 0
68     referencia = motion.FRAME_WORLD #Referencia estado inicial del robot al ↪
69             iniciar animacion
70                         #Vale 1
71     referencia = motion.FRAME_ROBOT #Referencia origen justo debajo del NAO ↪
72             entre los pies
73                         #Vale 2
74 if marcoRef.upper() == "TORSO":
75     referencia = motion.FRAME_TORSO
76     print "      Frame set to TORSO"
77 elif marcoRef.upper() == "ROBOT" or marcoRef.upper() == "ARRIBA":
78     referencia = motion.FRAME_ROBOT
79     print "      Frame set to ROBOT"
80 else:
81     error.abort("Did not receive a valid Reference Frame.", "Move")
82 ##Relacion coordenadas con marco de referencia
83     #True para usar coordenadas absolutas respecto al marco de referencia
84 absolutos = True
85 ##Grados de libertad a utilizar
86 #####Debe incluir un elemento por cada actuador a controlar
87 ##### AXIS_MASK_ALL controla XYZ y rotacion
88 ##### AXIS_MASK_VEL controla solo XYZ
89 axisMask = [ motion.AXIS_MASK_ALL, motion.AXIS_MASK_ALL, motion.AXIS_MASK_ALL ↪
90             , motion.AXIS_MASK_ALL, motion.AXIS_MASK_ALL, motion.AXIS_MASK_ALL ]
91 if marcoRef.upper() == "ARRIBA":
92     axisMask = [ motion.AXIS_MASK_VEL ]*2
93 elif marcoRef.upper() == "ROBOT":
94     axisMask = [ motion.AXIS_MASK_VEL ]*4

```

```

94     elif marcoRef.upper() == "TORSO":
95         axisMask = [ motion.AXIS_MASK_VEL ]*4
96
97 #-----
98 #Obtencion de la informacion del archivo CVS
99     print "-----"
100    print "      Starting adjustment of data..."
101    #Realiza ajuste de los datos de la grabacion
102    csvMocap.startAdjustData(coreo, offFile)
103
104   print "      Adjustment completed."
105   print "+++++"
106   ##Lista con vectores de las posiciones de los actuadores en orden ↔
107   correspondiente
108   ##al orden de los actuadores a utilizar
109   print "      Getting coordinates"
110   print "-----"
111   #Obtiene lista con coordenadas
112   listaCoordenadas = csvMocap.getCoordenadas(marcoRef)
113
114   print "      Getting times"
115   print "-----"
116   ##Lista de Tiempos
117   listaTiempos = csvMocap.getTiempos(marcoRef)
118
119   print "      Getting actuator names"
120   print "-----"
121   ##Lista de Actuadores en el orden a ser usadas
122   ###Orden Preferido "RArm", "RLeg", "LLeg", "LArm", "Torso", "Head"
123   if marcoRef.upper() == "ROBOT":
124       listaActuadores = ["RArm", "LArm", "Head"]
125   elif marcoRef.upper() == "TORSO":
126       listaActuadores = ["RArm", "RLeg", "LLeg", "LArm", "Head"]
127   elif marcoRef.upper() == "ARRIBA":
128       listaActuadores = ["RArm", "LArm", "Torso"]
129   listaActuadores = ["RArm", "LArm"]
130
131 #-----
132 #-----
133 #Control del movimiento del NAO -- Teleoperacion
134     -----
135     print "+++++"
136     print "Setting up movement conditions"
137     print "+++++"
138     #Preparativos iniciales para mover el NAO
139
140     ##Iniciando motores y activando rigidez para poder iniciar el movimiento
141     ##en el NAO, si la rigidez no esta puesta el Nao no se mueve
142     #motionProxy.wakeUp()
143     print "      Stiffness is ON"

```

```

144     StiffnessOn(motionProxy)
145
146     print "    NAO standing"
147     ##Llevando al NAO a una pose segura para moverse
148     postureProxy.goToPosture("StandInit", 0.5)
149     #postureProxy.goToPosture("Stand", 0.5) #Pose preferente
150
151     ##Una vez que esta en una posicion segura se puede inhabilitar el controlador
152     ##automatico de caidas, esto para que no restrinja ciertas posiciones del Nao
153     ### **Tener en cuenta que ahora corre peligro de caidas daninas, el Nao Debe
154     ##estar en un ambiente controlado y el usuario atento a caidas para que
155     ##no sufra danos evitables**
156     print "++++++++++++++++++++++++++++++++++++++"
157     print "++++++++++++++++++++++++++++++++++++++"
158     print "          Disabling Fall Manager      "
159     print "          WARNING:                  "
160     print "          NAO wont react automatically to falls.      "
161     print "Please take care of the NAO robot during the"
162     print "          execution of the teleoperation      "
163     print "++++++++++++++++++++++++++++++++++++++"
164     print "++++++++++++++++++++++++++++++++++++++"
165     time.sleep(3.0)
166     motionProxy.setFallManagerEnabled(False)
167
168     ##Habilita Balanceo de Cuerpo Completo Automatico
169     motionProxy.wbEnable(True)
170
171     ##Restriccion de soporte para las piernas
172     ###Condicion de operacion
173     estadoFijo = "Fixed" # Posicion Fija
174     estadoPlano = "Plane" # Permite desplazamiento sobre el plano
175     estadoLibre = "Free" # Libre movimiento en el espacio
176     ###Actuador de soporte
177     #### Legs usa ambas piernas, sin embargo se puede manejar la restriccion
178     #### para cada pierna de manera independiente
179     soportePiernas = "Legs" # Ambas piernas
180     soporteIzq = "LLeg" # Pierna izquierda
181     soporteDer = "RLeg" # Pierna derecha
182
183     ###Habilita soporte de ambas piernas con restricciones
184     motionProxy.wbFootState(estadoFijo, soportePiernas)
185     ###Para usar piernas con estados individuales
186     #motionProxy.wbFootState(estadoFijo, soporteDer)
187     #motionProxy.wbFootState(estadoLibre, soporteIzq)
188
189     ##Habilita balance del cuerpo sobre el soporte definido
190     soporteActivo = True
191     motionProxy.wbEnableBalanceConstraint(soporteActivo, soportePiernas)
192     print "    Balance constraints set"
193
194     #-----

```

```

195     print "++++++++++++++++++++++++++++++++++++++"
196     print "++++++++++++++++++++++++++++++++++++++"
197     print "NAO movement started ... "
198     #Inicio del movimiento
199
200     ##Tiempo de espera para iniciar movimiento
201     time.sleep(1.0)
202     ##Ejecucion de las posiciones obtenidas del archivo csvMocap, todo de un solo
203     #motionProxy.positionInterpolations(listaActuadores, referencia, ↵
204     #    listaCoordenadas, axisMask, listaTiempos, absolutos)
205     ##Ejecucion de las posiciones obtenidas del archivo csvMocap, por cuadros (30 ↵
206     #    FPS)
207     fps = 30
208     COM = [0.0 ,0.0 ,0.0]
209     listaTorsoX = list()
210     listaTorsoY = list()
211     balancePiernas = "centro"
212
213     #motion.wbEnableEffectorOptimization(effectName, isActive);
214
215     # Utiliza 3 actuadores: RArm, LArm y Torso
216     if marcoRef.upper() == "ARRIBA":
217         for i in range(0, len(listaCoordenadas[0]),fps):
218             #Calcula promedio de la posicion del torso deseada
219             #for item in listaCoordenadas[2][i:i+fps -1]:
220             #    listaTorsoX.append(item[0])
221             #    listaTorsoY.append(item[1])
222             #promTorsoX = np.mean(listaTorsoX)
223             #promTorsoY = np.mean(listaTorsoY)
224
225             #Obtiene posicion del COM general
226             #COM = motionProxy.getCOM("Body",referencia, True)
227             #Para apoyarse sobre alguna pierna se analiza donde se va a posicionarse
228             #al
229             #torso sobre el eje Y (izq y der)
230             #if promTorsoY < -0.025:
231                 #motionProxy.wbGoToBalance(soporteDer, 1.0)
232             #elif promTorsoY > 0.025:
233                 #motionProxy.wbGoToBalance(soporteDer, 1.0)
234             #else:
235                 #motionProxy.wbGoToBalance(soportePiernas, 1.0)
236
237             #Luego de evaluar el mejor apoyo para el posicionamiento deseado se
238             #realiza el movimiento del resto de los actuadores
239             motionProxy.positionInterpolations(listaActuadores, referencia, [←
240                 listaCoordenadas[0][i:i+fps -1], listaCoordenadas[1][i:i+fps -1]], ←
241                 axisMask, [ listaTiempos[0][0:len(listaCoordenadas[0][i:i+fps -1])]←
242                 ], listaTiempos[1][0:len(listaCoordenadas[1][i:i+fps -1])]], ←
243                 absolutos)
244
245             #Utiliza 4 actuadores: RArm, LArm, Torso y Head

```

```

239     elif marcoRef.upper() == "ROBOT":
240         for i in range(0,len(listaCoordenadas[0]),fps):
241             motionProxy.positionInterpolations(listaActuadores, referencia,[←
242                 listaCoordenadas[0][i:i+fps-1],listaCoordenadas[1][i:i+fps-1],←
243                 listaCoordenadas[2][i:i+fps-1],listaCoordenadas[3][i:i+fps-1] ],←
244                 axisMask, [listaTiempos[0][0:len(listaCoordenadas[0][i:i+fps-1])],←
245                 listaTiempos[1][0:len(listaCoordenadas[1][i:i+fps-1])],←
246                 listaTiempos[2][0:len(listaCoordenadas[2][i:i+fps-1])],←
247                 listaTiempos[3][0:len(listaCoordenadas[3][i:i+fps-1])]],absolutos←
248             )
249
250
251 #Utiliza 6 actuadores: RArm, RLeg, LLeg, LArm, Torso y Head
252 elif marcoRef.upper() == "TORSO":
253     for i in range(0,len(listaCoordenadas[0]),fps):
254         #Calcula promedio de la posicion del torso deseada
255         for item in listaCoordenadas[2][i:i+fps-1]:
256             listaTorsoX.append(item[0])
257             listaTorsoY.append(item[1])
258         promTorsoX = np.mean(listaTorsoX)
259         promTorsoY = np.mean(listaTorsoY)
260
261         #Obtiene posicion del COM general
262         COM = motionProxy.getCOM("Body",referencia, True)
263         #Para apoyarse sobre alguna pierna se analiza donde se va a posicionar←
264         #al
265         #torso sobre el eje Y (izq y der)
266         if promTorsoY < -0.03:
267             if balancePiernas != "der":
268                 motionProxy.wbGoToBalance(soporteDer, 0.5)
269                 balancePiernas = "der"
270             elif promTorsoY > 0.03:
271                 if balancePiernas != "izq":
272                     motionProxy.wbGoToBalance(soporteDer, 0.5)
273                     balancePiernas = "izq"
274             else:
275                 if balancePiernas != "centro":
276                     motionProxy.wbGoToBalance(soportePiernas, 0.5)
277                     balancePiernas = "centro"
278
279             motionProxy.positionInterpolations(listaActuadores, referencia,[←
280                 listaCoordenadas[0][i:i+fps-1],listaCoordenadas[1][i:i+fps-1],←
281                 listaCoordenadas[2][i:i+fps-1],listaCoordenadas[3][i:i+fps-1] ],←
282                 axisMask, [listaTiempos[0][0:len(listaCoordenadas[0][i:i+fps-1])],←
283                 listaTiempos[1][0:len(listaCoordenadas[1][i:i+fps-1])],←
284                 listaTiempos[2][0:len(listaCoordenadas[2][i:i+fps-1])],←
285                 listaTiempos[3][0:len(listaCoordenadas[3][i:i+fps-1])]],absolutos←
286             )
287
288         print "-----"
289         print "NAO movement ended. Resting NAO."

```

```
275     print "+++++++"  
276     print "+++++++"  
277     ##Tiempo de espera entre ultimo movimiento y estado de reposo del Nao, para  
278     ##agregar estabilidad  
279     time.sleep(2.0)  
280  
281     print "    NAO moving to posture: Crouch"  
282     print "-----"  
283     ##Posicion de reposo seguras para finalizar la accion  
284     postureProxy.goToPosture("Crouch", 0.5)  
285  
286     #Se habilita nuevamente el controlador automatico de caidas  
287     # **** IMPORTANTE REALIZAR ESTE PASO****#  
288     motionProxy.setFallManagerEnabled(True)  
289     ##Desactiva Balance de Cuerpo Completo **Debe ir al final del movimiento**  
290     motionProxy.wbEnable(False)  
291     print "*****"  
292     print "Fall Manager Enabled"  
293     print "*****"  
294  
295     motionProxy.rest() # Apaga los motores del Nao  
296     ## ** Si no se apagan es posible que se sobrecalienten aun estando en reposo**  
297     print "    NAO is resting."  
298     print "+++++++"  
299     print "Teleoperation ended."  
300     print "+++++++"  
301     print "+++++++"  
302  
303 #-----  
304 #-----  
305 if __name__ == "__main__":  
306     robotIp = "10.0.1.128" #Bato por red PrisNao  
307     #robotIp = "169.254.42.173" #Bato Local  
308     coreo = ""  
309     marcoRef = "ROBOT"  
310     offFile = "offsets.csv"  
311  
312     if len(sys.argv) == 4:  
313         robotIp = sys.argv[1]  
314         coreo = sys.argv[2]  
315         marcoRef = sys.argv[3]  
316         print "+++++++"  
317         print "Using robot IP:", robotIp  
318         print "Choreography file to read:", coreo  
319         print "Using reference plane:", marcoRef  
320         print "Using calibration data from default:", offFile  
321         print "-----"  
322         print "Initializing Teleoperation with Data from a MoCap recording"  
323         print "+++++++"  
324         time.sleep(1.5) #Tiempo para que el usuario lea las indicaciones  
325     elif len(sys.argv) == 5:
```

```
326     robotIp = sys.argv[1]
327     coreo   = sys.argv[2]
328     marcoRef = sys.argv[3]
329     offFile = sys.argv[4]
330     print "++++++"
331     print "Using robot IP:", robotIp
332     print "Choreography file to read:", coreo
333     print "Using reference plane:", marcoRef
334     print "Using calibration data from file:", offFile
335     print "-----"
336     print "Initializing Teleoperation with Data from a MoCap recording"
337     print "++++++"
338     time.sleep(1.5) #Tiempo para que el usuario lea las indicaciones
339 else:
340     error.abort("Expected 3 or 4 arguments on call.", "Move")
341
342 main(robotIp,coreo,marcoRef,offFile)
```

C.4.2. CSVMOCAPFunc.py

```
1 #####  
2 #Incluye funciones para la lectura del archivo CSV y ajuste a los datos de la  
3 #grabacion del movimiento capturado en MoCap que se quiere ejecutar.  
4 ##  
5 #Recibe los terminos del ajuste de calibracion del archivo CSV  
6 #.../Cal/Offsets/offsets.csv. Devuelve listas con los actuadores y vectores de  
7 #coordenadas y tiempos, Las coordenadas pueden ser obtenidas en los marcos de  
8 #referencia ROBOT y TORSO, segun se solicite.  
9 #####  
10 #Imports  
11 import csv  
12 import os  
13 from itertools import islice  
14 from os.path import dirname, abspath  
15 from copy import deepcopy  
16  
17 ##Custom  
18 import OffsetFileFunc as offset  
19 import ErrorFunc as error  
20  
21 -----  
22 -----  
23 #Globales  
24 coordenadasArriba = list()  
25 coordenadasROBOT = list()  
26 coordenadasTORSO = list()  
27 coordenadasCompletas = list()  
28 listaTiemposROBOT = list()  
29 listaTiemposTORSO = list()  
30 listaTiemposArriba = list()  
31 listaTiemposCompletos = list()  
32  
33 #Inicia el procesamiento de los archivos CSV necesarios para el ajuste de datos  
34 #Recibe como parametro el nombre del archivo CSV con las coordenadas a leer  
35 def startAdjustData(nombreArchivo):  
36     print "+++++++"  
37     print "Starting coordinates adjustment for", nombreArchivo  
38     #Coeficientes de Ajuste para valores del MoCap, considerando relacion lineal  
39     #entre datos del MoCap y datos del Nao  
40     ##Lectura del archivo con los parametros  
41     try:  
42         listaOffsets,degree,rotacion = offset.getOffsets()  
43     except Exception,e:  
44         error.abort("Failed to get offsets from file offset.csv, check file", None  
        , "CSVMOCAPFunc")  
45  
46     #Sin rotaciones, solo X,Y,Z  
47     if rotacion.lower() == "no":
```

```

48     offRArm = [[ ] for x in range(3)]
49     offRLeg = [[ ] for x in range(3)]
50     offLLeg = [[ ] for x in range(3)]
51     offLArm = [[ ] for x in range(3)]
52     offTorso = [[ ] for x in range(3)]
53     offHead = [[ ] for x in range(3)]
54
55
56 #RArm
57 for j in range(degree+1):
58     offRArm[0].append(round(float(listaOffsets[0][j]),4))
59     offRArm[1].append(round(float(listaOffsets[1][j]),4))
60     offRArm[2].append(round(float(listaOffsets[2][j]),4))
61 #RLeg
62 for j in range(degree+1):
63     offRLeg[0].append(round(float(listaOffsets[3][j]),4))
64     offRLeg[1].append(round(float(listaOffsets[4][j]),4))
65     offRLeg[2].append(round(float(listaOffsets[5][j]),4))
66 #LLeg
67 for j in range(degree+1):
68     offLLeg[0].append(round(float(listaOffsets[6][j]),4))
69     offLLeg[1].append(round(float(listaOffsets[7][j]),4))
70     offLLeg[2].append(round(float(listaOffsets[8][j]),4))
71
72 #LArm
73 for j in range(degree+1):
74     offLArm[0].append(round(float(listaOffsets[9][j]),4))
75     offLArm[1].append(round(float(listaOffsets[10][j]),4))
76     offLArm[2].append(round(float(listaOffsets[11][j]),4))
77
78 #Torso
79 for j in range(degree+1):
80     offTorso[0].append(round(float(listaOffsets[12][j]),4))
81     offTorso[1].append(round(float(listaOffsets[13][j]),4))
82     offTorso[2].append(round(float(listaOffsets[14][j]),4))
83
84 #Head
85 for j in range(degree+1):
86     offHead[0].append(round(float(listaOffsets[15][j]),4))
87     offHead[1].append(round(float(listaOffsets[16][j]),4))
88     offHead[2].append(round(float(listaOffsets[17][j]),4))
89
90 #Conrotaciones , X,Y,Z,wX,wY,wZ
91 elif rotacion.lower() == "yes":
92     offRArm = [[ ] for x in range(6)]
93     offRLeg = [[ ] for x in range(6)]
94     offLLeg = [[ ] for x in range(6)]
95     offLArm = [[ ] for x in range(6)]
96     offTorso = [[ ] for x in range(6)]
97     offHead = [[ ] for x in range(6)]
98

```

```

99
100    #RArm
101    for j in range(degree+1):
102        offRArm[0].append(round(float(listaOffsets[0][j]),4))
103        offRArm[1].append(round(float(listaOffsets[1][j]),4))
104        offRArm[2].append(round(float(listaOffsets[2][j]),4))
105        offRArm[3].append(round(float(listaOffsets[3][j]),4))
106        offRArm[4].append(round(float(listaOffsets[4][j]),4))
107        offRArm[5].append(round(float(listaOffsets[5][j]),4))
108
109    #RLeg
110    for j in range(degree+1):
111        offRLeg[0].append(round(float(listaOffsets[6][j]),4))
112        offRLeg[1].append(round(float(listaOffsets[7][j]),4))
113        offRLeg[2].append(round(float(listaOffsets[8][j]),4))
114        offRLeg[3].append(round(float(listaOffsets[9][j]),4))
115        offRLeg[4].append(round(float(listaOffsets[10][j]),4))
116        offRLeg[5].append(round(float(listaOffsets[11][j]),4))
117
118    #LLeg
119    for j in range(degree+1):
120        offLLeg[0].append(round(float(listaOffsets[12][j]),4))
121        offLLeg[1].append(round(float(listaOffsets[13][j]),4))
122        offLLeg[2].append(round(float(listaOffsets[14][j]),4))
123        offLLeg[3].append(round(float(listaOffsets[15][j]),4))
124        offLLeg[4].append(round(float(listaOffsets[16][j]),4))
125        offLLeg[5].append(round(float(listaOffsets[17][j]),4))
126
127    #LArm
128    for j in range(degree+1):
129        offLArm[0].append(round(float(listaOffsets[18][j]),4))
130        offLArm[1].append(round(float(listaOffsets[19][j]),4))
131        offLArm[2].append(round(float(listaOffsets[20][j]),4))
132        offLArm[3].append(round(float(listaOffsets[21][j]),4))
133        offLArm[4].append(round(float(listaOffsets[22][j]),4))
134        offLArm[5].append(round(float(listaOffsets[23][j]),4))
135
136    #Torso
137    for j in range(degree+1):
138        offTorso[0].append(round(float(listaOffsets[24][j]),4))
139        offTorso[1].append(round(float(listaOffsets[25][j]),4))
140        offTorso[2].append(round(float(listaOffsets[26][j]),4))
141        offTorso[3].append(round(float(listaOffsets[27][j]),4))
142        offTorso[4].append(round(float(listaOffsets[28][j]),4))
143        offTorso[5].append(round(float(listaOffsets[29][j]),4))
144
145    #Head
146    for j in range(degree+1):
147        offHead[0].append(round(float(listaOffsets[30][j]),4))
148        offHead[1].append(round(float(listaOffsets[31][j]),4))
149        offHead[2].append(round(float(listaOffsets[32][j]),4))

```

```

150         offHead[5].append(round(float(listaOffsets[35][j]),4))
151     else:
152         error.abort("Expected \"yes\" or \"no\" on rotation statement.", rotacion,←
153                     "CSVMOCAPFunc.py")
154     #←
155     #←
156     print "      Reading choreography file"
157     #Obencion de directorio base
158     rootDir = dirname(dirname(abspath(__file__)))
159     #Declarando directorio para abrir archivo CSV
160     ##Nombre del archivo CSV a leer
161     archivo = os.path.join(rootDir, "Ver5/Choreography/")
162     archivo = os.path.join(archivo, nombreArchivo)
163
164     #Creando objeto con contenido del archivo CSV
165     ##Abriendo archivo
166     try:
167         f = open(archivo, 'rt')
168     except IOError:
169         error.abort("Archivo no encontrado, revise nombre de la rutina a ejecutar.←
170                     ", None, "CSVMOCAPFunc")
171     ##Obteniendo datos completos y cerrando archivo
172     reader = csv.reader(f)
173     filasIniciales = [r for r in reader]
174     f.close()
175     #←
176     #←
177     #Extrayendo informacion importante del archivo
178     ##Obencion del orden de aparicion de los marcadores(actuadores)
179     filasActuadores = filasIniciales[3]
180     ###Remueve primeros dos espacios siempre en blanco
181     filasActuadores.remove(' ')
182     filasActuadores.remove(' ')
183     j = 0
184     listaActuadores = [None]*6 #Se trabaja con 6 marcadores
185     if rotacion.lower() == "no":
186         for i, item in enumerate(filasActuadores):
187             #Se repite el nombre del marcador 3 veces(XYZ)
188             if i==0 or i==3 or i==6 or i==9 or i==12 or i==15:
189                 listaActuadores[j] = str(item)
190                 j+=1

```



```

235         contXYZ+=1
236     elif contXYZ == 2 :
237         try :
238             coordenadas[contXYZ] = float(filasCoordenadas[i].pop←
239                                         (0))
240         except ValueError:
241             print "Cell is not a valid value"
242         contXYZ=0
243         #trioTemp[contXYZ] = coordenadas[contXYZ]
244         #Se tienen XYZ+rot para un actuador en un cuadro ←
245         #especifico
246         ### Z e Y estan invertidos en el marco de referencia del ←
247         #MoCap
248         ### Rotaciones en 0.0 , X*-1
249         #####Sin importar el orden de los actuadores en el archivo ←
250         #aqui
251         #####se acomodan en el orden preferente y se aplica la ←
252         #correccion
253         #####con los offsets
254         if listaActuadores[contActuador] == "RArm":
255             actuador[0].append([ round((coordenadas[0]**2)*offRArm←
256                                 [0][0] +(-1)*coordenadas[0]*offRArm[0][1]+offRArm←
257                                 [0][2], 2),
258                             round((coordenadas[2]**2)*offRArm←
259                                 [1][0] +      coordenadas[2]*←
260                                 offRArm[1][1]+offRArm[1][2], ←
261                                 2),
262                             round((coordenadas[1]**2)*offRArm←
263                                 [2][0] +      coordenadas[1]*←
264                                 offRArm[2][1]+offRArm[2][2], ←
265                                 2),
266                             0.0 , 0.0 , 0.0]) #Se pone ←
267             rotaciones en 0 ya que no se ←
268             van a utilizar
269         elif listaActuadores[contActuador] == "RLeg":
270             actuador[1].append([ round((coordenadas[0]**2)*offRArm←
271                                 [0][0] +(-1)*coordenadas[0]*offRArm[0][1]+offRArm←
272                                 [0][2], 2),
273                             round((coordenadas[2]**2)*offRArm←
274                                 [1][0] +      coordenadas[2]*←
275                                 offRArm[1][1]+offRArm[1][2], ←
276                                 2),
277                             round((coordenadas[1]**2)*offRArm←
278                                 [2][0] +      coordenadas[1]*←
279                                 offRArm[2][1]+offRArm[2][2], ←
280                                 2),
281                             0.0 , 0.0 , 0.0])
282         elif listaActuadores[contActuador] == "LLeg":
283             actuador[2].append([ round((coordenadas[0]**2)*offRArm←
284                                 [0][0] +(-1)*coordenadas[0]*offRArm[0][1]+offRArm←
285                                 [0][2], 2),
286

```

```

261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
    round((coordenadas[2]**2)*offRArm<-
           [1][0] +      coordenadas[2]*<-
           offRArm[1][1]+offRArm[1][2], <-
           2),
    round((coordenadas[1]**2)*offRArm<-
           [2][0] +      coordenadas[1]*<-
           offRArm[2][1]+offRArm[2][2], <-
           2),
           0.0, 0.0, 0.0])
elif listaActuadores[contActuador] == "LArm":
    actuador[3].append([round((coordenadas[0]**2)*offRArm<-
                               [0][0] +(-1)*coordenadas[0]*offRArm[0][1]+offRArm<-
                               [0][2], 2),
                         round((coordenadas[2]**2)*offRArm<-
                               [1][0] +      coordenadas[2]*<-
                               offRArm[1][1]+offRArm[1][2], <-
                               2),
                         round((coordenadas[1]**2)*offRArm<-
                               [2][0] +      coordenadas[1]*<-
                               offRArm[2][1]+offRArm[2][2], <-
                               2),
                         0.0, 0.0, 0.0])
elif listaActuadores[contActuador] == "Torso":
    actuador[4].append([round((coordenadas[0]**2)*offRArm<-
                               [0][0] +(-1)*coordenadas[0]*offRArm[0][1]+offRArm<-
                               [0][2], 2),
                         round((coordenadas[2]**2)*offRArm<-
                               [1][0] +      coordenadas[2]*<-
                               offRArm[1][1]+offRArm[1][2], <-
                               2),
                         round((coordenadas[1]**2)*offRArm<-
                               [2][0] +      coordenadas[1]*<-
                               offRArm[2][1]+offRArm[2][2], <-
                               2),
                         0.0, 0.0, 0.0])
elif listaActuadores[contActuador] == "Head":
    actuador[5].append([round((coordenadas[0]**2)*offRArm<-
                               [0][0] +(-1)*coordenadas[0]*offRArm[0][1]+offRArm<-
                               [0][2], 2),
                         round((coordenadas[2]**2)*offRArm<-
                               [1][0] +      coordenadas[2]*<-
                               offRArm[1][1]+offRArm[1][2], <-
                               2),
                         round((coordenadas[1]**2)*offRArm<-
                               [2][0] +      coordenadas[1]*<-
                               offRArm[2][1]+offRArm[2][2], <-
                               2),
                         0.0, 0.0, 0.0])
contActuador+=1
if contActuador == 6:
    contActuador = 0

```

```

282     except Exception,e:
283         error.abort("Check file data, not able to read all of it", None, "↔
284                     CSVMOCAFFunc", "Move")
285
286 #Ajuste con rotaciones
287 if rotacion.lower() == "yes":
288     coordenadas = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] #Incluye wX,wY,wZ,W↔
289                 ,X,Y,Z,Error
290     #try:
291     for i, item in enumerate(filasCoordenadas):
292         #contActuador = 1 #Reinicia contador de actuador cada vez que carga ↔
293         #fila nueva
294         for contTrios in range(0,48) :#(6 DoF+2 extra data) * 6 actuadores
295             if contXYZ < 7 :
296                 coordenadas[contXYZ] = float(filasCoordenadas[i].pop(0))
297                 contXYZ+=1
298             elif contXYZ == 7 :
299                 coordenadas[contXYZ] = float(filasCoordenadas[i].pop(0))
300                 contXYZ=0
301                 #Orden datos en CSV (wX,wY,wZ,W,X,Y',Z',Error), con Y' y Z' ↔
302                 #cambiados
303                 #Orden necesario de los datos (X,Z',Y',wX,wY,wZ), con Y'=Z y Z↔
304                 '=Y
305             if listaActuadores[contActuador] == "RArm":
306                 actuador[0].append([round((coordenadas[4]**2)*offRArm↔
307                                 [0][0] + coordenadas[4]* offRArm[0][1] + offRArm↔
308                                 [0][2], 2),
309                                 round((coordenadas[6]**2)*offRArm↔
310                                     [1][0] + coordenadas[6]* offRArm↔
311                                     [1][1] + offRArm[1][2], 2),
312                                     round((coordenadas[5]**2)*offRArm↔
313                                         [2][0] + coordenadas[5]* offRArm↔
314                                         [2][1] + offRArm[2][2], 2),
315                                         round((coordenadas[0]**2)*offRArm↔
316                                             [3][0] + coordenadas[0]* offRArm↔
317                                             [3][1] + offRArm[3][2], 2),
318                                             round((coordenadas[1]**2)*offRArm↔
319                                                 [4][0] + coordenadas[1]* offRArm↔
320                                                 [4][1] + offRArm[4][2], 2),
321                                                 round((coordenadas[2]**2)*offRArm↔
322                                                     [5][0] + coordenadas[2]* offRArm↔
323                                                     [5][1] + offRArm[5][2], 2),
324                                                     ])
325             elif listaActuadores[contActuador] == "RLeg":
326                 actuador[1].append([round((coordenadas[4]**2)*offRLeg↔
327                                 [0][0] + coordenadas[4]* offRLeg[0][1] + offRLeg↔
328                                 [0][2], 2),
329                                 round((coordenadas[6]**2)*offRLeg↔
330                                     [1][0] + coordenadas[6]* offRLeg↔
331                                     [1][1] + offRLeg[1][2], 2),
332                                     ])

```

```

311     round((coordenadas[5]**2)*offRLeg<-
312         [2][0] + coordenadas[5]* offRLeg<-
313             [2][1] + offRLeg[2][2], 2),
314     round((coordenadas[0]**2)*offRLeg<-
315         [3][0] + coordenadas[0]* offRLeg<-
316             [3][1] + offRLeg[3][2], 2),
317     round((coordenadas[1]**2)*offRLeg<-
318         [4][0] + coordenadas[1]* offRLeg<-
319             [4][1] + offRLeg[4][2], 2),
320     round((coordenadas[2]**2)*offRLeg<-
321         [5][0] + coordenadas[2]* offRLeg<-
322             [5][1] + offRLeg[5][2], 2),
323     ))
324 elif listaActuadores[contActuador] == "LLeg":
325     actuador[2].append([round((coordenadas[4]**2)*offLLeg<-
326         [0][0] + coordenadas[4]* offLLeg[0][1] + offLLeg<-
327             [0][2], 2),
328         round((coordenadas[6]**2)*offLLeg<-
329             [1][0] + coordenadas[6]* offLLeg<-
330                 [1][1] + offLLeg[1][2], 2),
331         round((coordenadas[5]**2)*offLLeg<-
332             [2][0] + coordenadas[5]* offLLeg<-
333                 [2][1] + offLLeg[2][2], 2),
334         round((coordenadas[0]**2)*offLLeg<-
335             [3][0] + coordenadas[0]* offLLeg<-
336                 [3][1] + offLLeg[3][2], 2),
337         round((coordenadas[1]**2)*offLLeg<-
338             [4][0] + coordenadas[1]* offLLeg<-
339                 [4][1] + offLLeg[4][2], 2),
340         round((coordenadas[2]**2)*offLLeg<-
341             [5][0] + coordenadas[2]* offLLeg<-
342                 [5][1] + offLLeg[5][2], 2),
343         ))
344 elif listaActuadores[contActuador] == "LArm":
345     actuador[3].append([round((coordenadas[4]**2)*offLArm<-
346         [0][0] + coordenadas[4]* offLArm[0][1] + offLArm<-
347             [0][2], 2),
348         round((coordenadas[6]**2)*offLArm<-
349             [1][0] + coordenadas[6]* offLArm<-
350                 [1][1] + offLArm[1][2], 2),
351         round((coordenadas[5]**2)*offLArm<-
352             [2][0] + coordenadas[5]* offLArm<-
353                 [2][1] + offLArm[2][2], 2),
354         round((coordenadas[0]**2)*offLArm<-
355             [3][0] + coordenadas[0]* offLArm<-
356                 [3][1] + offLArm[3][2], 2),
357         round((coordenadas[1]**2)*offLArm<-
358             [4][0] + coordenadas[1]* offLArm<-
359                 [4][1] + offLArm[4][2], 2),
360         round((coordenadas[2]**2)*offLArm<-
361             [5][0] + coordenadas[2]* offLArm<-
362                 [5][1] + offLArm[5][2], 2)
363     ])

```

```

331                               [5][1] + offLArm[5][2], 2),
332                               ])
333               elif listaActuadores[contActuador] == "Torso":
334                   actuador[4].append([round((coordenadas[4]**2)*offTorso<-
335                               [0][0] + coordenadas[4]*offTorso[0][1] + offTorso<-
336                               [0][2], 2),
337                               round((coordenadas[6]**2)*offTorso<-
338                               [1][0] + coordenadas[6]*offTorso<-
339                               [1][1] + offTorso[1][2], 2),
340                               round((coordenadas[5]**2)*offTorso<-
341                               [2][0] + coordenadas[5]*offTorso<-
342                               [2][1] + offTorso[2][2], 2),
343                               round((coordenadas[0]**2)*offTorso<-
344                               [3][0] + coordenadas[0]*offTorso<-
345                               [3][1] + offTorso[3][2], 2),
346                               round((coordenadas[1]**2)*offTorso<-
347                               [4][0] + coordenadas[1]*offTorso<-
348                               [4][1] + offTorso[4][2], 2),
349                               round((coordenadas[2]**2)*offTorso<-
350                               [5][0] + coordenadas[2]*offTorso<-
351                               [5][1] + offTorso[5][2], 2),
352                               ])
353               elif listaActuadores[contActuador] == "Head":
354                   actuador[5].append([round((coordenadas[4]**2)*offHead<-
355                               [0][0] + coordenadas[4]*offHead[0][1] + offHead<-
356                               [0][2], 2),
357                               round((coordenadas[6]**2)*offHead<-
358                               [1][0] + coordenadas[6]*offHead<-
359                               [1][1] + offHead[1][2], 2),
360                               round((coordenadas[5]**2)*offHead<-
361                               [2][0] + coordenadas[5]*offHead<-
362                               [2][1] + offHead[2][2], 2),
363                               round((coordenadas[0]**2)*offHead<-
364                               [3][0] + coordenadas[0]*offHead<-
365                               [3][1] + offHead[3][2], 2),
366                               round((coordenadas[1]**2)*offHead<-
367                               [4][0] + coordenadas[1]*offHead<-
368                               [4][1] + offHead[4][2], 2),
369                               round((coordenadas[2]**2)*offHead<-
370                               [5][0] + coordenadas[2]*offHead<-
371                               [5][1] + offHead[5][2], 2),
372                               ])
373               contActuador+=1
374               if contActuador == 6:
375                   contActuador = 0
376               # except Exception ,e:
377               #     error.abort("Check file data, not able to read all of it", None, "←
378               #             CSVMOCAFFunc", "Move")
379               #En este punto ya se tienen los vectores de posiciones XYZ+rotacion(0.0) para ←
380               #cada

```

```

355     #actuador independiente , en el orden segun el archivo CSV
356
357     ##Generando Vector completo como lista de vectores para cada actuador
358     global coordenadasCompletas
359     coordenadasCompletas = [actuador[0], actuador[1], actuador[2], actuador[3], ←
360         actuador[4], actuador[5]]
360     global coordenadasArriba
361     coordenadasArriba = [actuador[0], actuador[3], actuador[4]]
362     global coordenadasROBOT
363     coordenadasROBOT = [actuador[0], actuador[3], actuador[4], actuador[5]]
364
365     ##Si los datos obtenidos vienen con referencia al TORSO no es necesario este
366     ##proceso de cambio de referencia
367     listaDeActuadores = deepcopy(coordenadasCompletas)
368     ## RArm
369     for i, item in enumerate(listaDeActuadores[0]):
370         for j, item2 in enumerate(listaDeActuadores[0][j]):
371             listaDeActuadores[0][i][j] = round((listaDeActuadores[0][i][j] - ←
372                 listaDeActuadores[4][i][j]), 2)
372     ## RLeg
373     for i, item in enumerate(listaDeActuadores[1]):
374         for j, item2 in enumerate(listaDeActuadores[1][j]):
375             listaDeActuadores[1][i][j] = round((listaDeActuadores[1][i][j] - ←
376                 listaDeActuadores[4][i][j]), 2)
376     ## LLeg
377     for i, item in enumerate(listaDeActuadores[2]):
378         for j, item2 in enumerate(listaDeActuadores[2][j]):
379             listaDeActuadores[2][i][j] = round((listaDeActuadores[2][i][j] - ←
380                 listaDeActuadores[4][i][j]), 2)
380     ## LArm
381     for i, item in enumerate(listaDeActuadores[3]):
382         for j, item2 in enumerate(listaDeActuadores[3][j]):
383             listaDeActuadores[3][i][j] = round((listaDeActuadores[3][i][j] - ←
384                 listaDeActuadores[4][i][j]), 2)
384     ## Head
385     for i, item in enumerate(listaDeActuadores[5]):
386         for j, item2 in enumerate(listaDeActuadores[5][j]):
387             listaDeActuadores[5][i][j] = round((listaDeActuadores[5][i][j] - ←
388                 listaDeActuadores[4][i][j]), 2)
389     ## TORSO
390     for i, item in enumerate(listaDeActuadores[4]):
391         for j, item2 in enumerate(listaDeActuadores[4][j]):
392             listaDeActuadores[4][i][j] = round((listaDeActuadores[4][i][j] - ←
393                 listaDeActuadores[4][i][j]), 2)
393     #Genera vector con datos respecto al TORSO
394     global coordenadasTORSO
394     coordenadasTORSO = [listaDeActuadores[0], listaDeActuadores[1], ←
395         listaDeActuadores[2], listaDeActuadores[3], listaDeActuadores[5]]
396     #←
-----←

```

```

397 #Base de tiempo predeterminado para cada vector de la animacion
398 ##Debe ser mayor a 20 ms (tiempo que dura en resolver el balance de cuerpo ↔
399      completo)
400 ##y dar al menos 30 ms entre cambios
401 ##coef depende de los cuadros por segundo de la animacion en Motive en el ↔
402      momento
403 ##de exportar los datos
404 #Se maneja un vector de tiempos independiente para cada actuador, con longitud
405 #correspondiente a la lista con coordenadas respectivo al actuador
406 coef = 0.05
407 #coef = 0.1
408 global listaTiemposROBOT
409 listaTiemposROBOT = [None]*len(coordenadasROBOT)
410 for i in range(len(coordenadasROBOT)):
411     listaTiemposROBOT[i] = [round(coef*(j+1),2) for j in range(len(↔
412         coordenadasROBOT[i]))]
413 global listaTiemposCompletos
414 listaTiemposCompletos = [None]*len(coordenadasCompletas)
415 for i in range(len(coordenadasROBOT)):
416     listaTiemposCompletos[i] = [round(coef*(j+1),2) for j in range(len(↔
417         coordenadasCompletas[i]))]
418 global listaTiemposArriba
419 listaTiemposArriba = [None]*len(coordenadasArriba)
420 for i in range(len(coordenadasArriba)):
421     listaTiemposArriba[i] = [round(coef*(j+1),2) for j in range(len(↔
422         coordenadasArriba[i]))]
423
424 print "Coordinates generated. Ready for movement."
425 print "++++++++++++++++++++++++++++++++++++++"
426
427 -----
428 -----
429 #Interfaz de extraccion de datos
430
431 ##Devuelve posiciones X,Y,Z+rot en orden correspondiente a los actuadores ↔
432      obtenidos
433 ##segun el marco de referencia
434 def getCoordenadas(frame):
435     if frame.upper() == "ROBOT":
436         return coordenadasROBOT
437     elif frame.upper() == "TORSO":
438         return coordenadasTORSO
439     elif frame.upper() == "ARRIBA":
440         return coordenadasArriba

```

```
440     else :
441         error.abort("Did not receive a valid reference frame.", None, "←
442                         CSVMOCAFFunc")
442
443 ##Devuelve lista de Tiempos del movimiento
444 ##según el marco de referencia
445 def getTiempos(frame):
446     if frame.upper() == "ROBOT":
447         return listaTiemposROBOT
448     elif frame.upper() == "TORSO":
449         return listaTiemposTORSO
450     elif frame.upper() == "ARRIBA":
451         return listaTiemposArriba
452     else:
453         error.abort("Did not receive a valid reference frame.", None, "←
453                         CSVMOCAFFunc")
```

Bibliografía

- [1] A. Adli. Human motion tracking for assisting balance training and control of a humanoid robot. Technical report, Universidad de Sur Florida, USA, 2012.
- [2] ALDEBARAN. Almotion. URL: <http://doc.aldebaran.com/2-1/naoqi/motion/almotion.html>.
- [3] ALDEBARAN. Alrobotposture. URL: <http://doc.aldebaran.com/2-1/naoqi/motion/alrobotposture.html>.
- [4] ALDEBARAN. Cartesian control. URL: <http://doc.aldebaran.com/1-14/naoqi/motion/control-cartesian.html>.
- [5] ALDEBARAN. Choregraphe suite installation. URL: <http://doc.aldebaran.com/1-14/software/installing.html>.
- [6] ALDEBARAN. Effector and chain definitions. URL: <http://doc.aldebaran.com/2-/family/robots/bodyparts.html>.
- [7] ALDEBARAN. Fall manager. URL: <http://doc.aldebaran.com/1-14/naoqi/motion/reflexes-fall-manager.html#fall-manager>.
- [8] ALDEBARAN. Fall manager api. URL: <http://doc.aldebaran.com/1-14/naoqi/motion/reflexes-fall-manager-api.html>.
- [9] ALDEBARAN. Joints. URL: http://doc.aldebaran.com/2-1/family/robots/joints_robot.html#robot-joints-v4-left-arm-joints.
- [10] ALDEBARAN. Masses. URL: http://doc.aldebaran.com/2-1/family/robots/masses_robot.html.
- [11] ALDEBARAN. Parallel tasks - making nao move and speak. URL: http://doc.aldebaran.com/1-14/dev/python/making_nao_move.html.
- [12] ALDEBARAN. Predefined postures. URL: http://doc.aldebaran.com/2-1/family/robots/postures_robot.html#robot-postures.
- [13] ALDEBARAN. Python sdk install guide. URL: http://doc.aldebaran.com/1-14/dev/python/install_guide.html.

- [14] ALDEBARAN. Whole body control. URL: <http://doc.aldebaran.com/2-1/naoqi/motion/control-wholebody.html>.
- [15] ALDEBARAN. Whole body control api. URL: <http://doc.aldebaran.com/2-1/naoqi/motion/control-wholebody-api.html>.
- [16] K. Atkinson. *An introduction to numerical analysis*. John Wiley and Sons, 1988.
- [17] K. Capek. *R.U.R (Rossum's Universal Robots)*. Dover Publications, Incorporated, 2001.
- [18] M Castresana. Diseño e implementación de un algoritmo para la corrección de glitches de mocap basado en goniometría. Technical report, Escuela de Ingeniería Eléctrica, Universidad de Costa Rica, Costa Rica, 2017.
- [19] National Research Council. *Virtual Reality: Scientific and Technological Challenges*. The National Academies Press, 1995.
- [20] A. De Luca. *Robots with kinematic redundancy*.
- [21] G. Doetsch, K. Lindberg. Canadarm. *The Canadian Encyclopedia*, Marzo 2015. URL: <http://www.thecanadianencyclopedia.ca/en/article/canadarm/>.
- [22] RoboCup Federation. Robocup. URL: <http://www.robocup.org/>.
- [23] B. Gates. A robot in every home. *Scientific American*, Enero 2007. URL: <https://www.scientificamerican.com/article/a-robot-in-every-home/>.
- [24] J. Jeppson. *It's Been a Good Life*. Prometheus Books, second edition, 2002.
- [25] C. Liu Y. Wang C. Jin, S. Dai. Motion imitation based on sparsely sampled correspondence. *ASME JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING*, Junio 2017. URL: <http://homepage.tudelft.nl/h05k3/pubs/JCISEMotionImitation.pdf>.
- [26] B. S Oh. Kim, S. Kim. Stable whole-body motion generation for humanoid robots to imitate human motionss. Technical report, en Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2009.
- [27] F. Bennewitz M. Koenemann, J. Burget. Real-time imitation of human whole-body motions by humanoids. Technical report, Universidad de Freiburg, Alemania, 2012.
- [28] M. Li Z. Chen C. Lei, J. Song. Whole-body humanoid robot imitation with pose similarity evaluation. *Signal Processing*, 108:136–146, 2015.
- [29] H. Markel. Science diction: The origin of the word ‘robot’, Abril 2011. URL: <https://www.sciencefriday.com/segments/science-diction-the-origin-of-the-word-robot/>.
- [30] S. Meredith, M. Maddock. Motion capture file formats explained. Technical report, Departamento de Ciencias Computacionales, Universidad de Sheffield, Inglaterra.

- [31] Microsoft. Kinect sensor for xbox one. URL: <https://www.microsoft.com/en-us/store/d/kinect-sensor-for-xbox-one/91hq5578vksc?activetab=pivot%3aoverviewtab>.
- [32] Podobnik J. Mihelj M. *Teleoperation. En: Haptics for Virtual Reality and Teleoperation.* Springer, Dordrecht, 2012.
- [33] G. Niemeyer. *Telerobotics.* Springer-Verlag, 2008.
- [34] NIST. Capítulo 4. e-handbook of statistical methods. URL: <http://www.itl.nist.gov/div898/handbook/pmd/section1/pmd141.htm>.
- [35] Elberly College of Science. Regression methods. URL: <https://onlinecourses.science.psu.edu/stat501/node/250>.
- [36] OptiTrack. Data export: Bvh. URL: http://v110.wiki.optitrack.com/index.php?title=Data_Export:_BVH.
- [37] OptiTrack. Data export: Csv. URL: http://wiki.optitrack.com/index.php?title=Data_Export:_CSV.
- [38] OptiTrack. Natnet api user's guide. URL: <https://optitrack.com/public/documents/natnet-api-user-guide-2.10.0.pdf>.
- [39] OptiTrack. Natnet: Sample projects. URL: http://v110.wiki.optitrack.com/index.php?title=NatNet:_Sample_Projects#NatNet_Sample_Projects.
- [40] OptiTrack. Natnet sdk. URL: <http://optitrack.com/products/natnet-sdk/>.
- [41] OptiTrack. Natnet sdk, wiki. URL: http://v110.wiki.optitrack.com/index.php?title=NatNet_SDK.
- [42] OptiTrack. Optitrack documentation wiki. URL: http://v110.wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki.
- [43] OptiTrack. Rigid body tracking. URL: http://v110.wiki.optitrack.com/index.php?title=Rigid_Body_Tracking.
- [44] D. Nakamura Y. Ott, C. Lee. Motion capture based human motion recognition and imitation by direct marker control. Technical report, Departamento de Mecano-Informáticos. Universidad de Tokio, Japón., 2008.
- [45] PRIS-Lab. *Manual del Mocap*, 2017.
- [46] ALDEBARAN robotics. *NAO User Guide*. 2012.
- [47] ROS.org. Aldebaran nao. URL: <http://wiki.ros.org/nao>.
- [48] T. Sheridan. Telerobotics. *Automatica*, 25:487–507.
- [49] T. Sheridan. *Telerobotics, Automation, and Human Supervisory Control.* The MIT Press, 1992.

- [50] SoftBank. Whos nao? URL: <https://www.ald.softbankrobotics.com/en/robots/nao>.
- [51] C Solano. Implementación y validación de un algoritmo de odometría visual monocular para la ubicación espacial del robothumanoide nao. Technical report, Escuela de Ingeniería Eléctrica, Universidad de Costa Rica, Costa Rica, 2017.
- [52] A. Ratanasena E. Stanton, C. Bogdanovych. Teleoperation of a humanoid robot using full-body motion capture, example movements, and machine learning. Technical report, Universidad de Tecnología, Sydney, Australia., 2012.
- [53] Suitable Technologies. Beam. *Página oficial del producto Beam*. URL: <https://suitabletech.com/products/beam>.