



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

TELEOPERACIÓN DEL ROBOT ICUB EN TIEMPO REAL

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA

FABIANA FERNANDA ALFARO GALLARDO

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

PROFESOR CO-GUÍA:
FRANCISCO LEIVA CASTRO

MIEMBROS DE LA COMISIÓN:
FRANCISCO RIVERA SERRANO
MARÍA JOSÉ ESCOBAR SILVA

Este proyecto ha sido parcialmente financiado por Proyecto ANID-PIA AFB220002

SANTIAGO DE CHILE
2024

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA
POR: FABIANA FERNANDA ALFARO GALLARDO
FECHA: 2024
PROF. GUÍA: JAVIER RUIZ DEL SOLAR SAN MARTÍN

TELEOPERACIÓN DEL ROBOT ICUB EN TIEMPO REAL

La teleoperación a través de la interacción humano-robot es uno de los desafíos de la ingeniería moderna. Trabajos previos han conseguido teleoperar robots humanoides, los cuales funcionan como avatares del operador humano gracias a la tecnología de captura de movimiento. Esta tecnología emplea sensores inerciales colocados en el cuerpo del operador para recabar información cinemática del usuario humano.

Este trabajo presenta un método basado en la tecnología de visión computacional y robótica para lograr la teleoperación de un robot humanoide iCub versión 2.7, sin la necesidad de sensores cinemáticos adheridos al cuerpo del operador humano y utilizando solo una cámara web para detectar sus movimientos. A lo largo del desarrollo de este proyecto, se emplearán algoritmos de estimación de postura, aproximaciones matemáticas y desarrollo de *software* necesario para lograr eliminar el uso de sensores cinemáticos, y generar por computadora la información requerida para el sistema de teleoperación.

El sistema desarrollado se valida utilizando visualizadores de movimiento del robot, simuladores que consideran las propiedades físicas del robot y su interacción con el entorno, y finalmente se valida la solución propuesta en un robot iCub v2.7 real facilitado por el Centro Avanzado de Ingeniería Eléctrica y Electrónica (AC3E).

Agradecimientos

Agradezco al profesor Javier Ruiz del Solar San Martín por brindarme la oportunidad de realizar este trabajo de título y por guiar el desarrollo de esta investigación.

Asimismo, expreso mi agradecimiento a Francisco Leiva Castro por su invaluable apoyo, constante acompañamiento y orientación durante todo mi trabajo y proceso de titulación, demostrando una disposición y compromiso excepcionales con sus alumnos.

Agradezco a la profesora María José Escobar Silva, a Miguel Rojas y al Centro Avanzado de Ingeniería Eléctrica y Electrónica (AC3E) por brindar y facilitar el acceso al robot iCub, recurso fundamental para la ejecución de este trabajo.

Quiero agradecer también a Rodrigo Navarro por su colaboración en el desarrollo de esta memoria y por fomentar espacios de diálogo enriquecedores para ampliar mis conocimientos en el ámbito de la robótica y este proyecto en particular.

A modo personal, quiero agradecer a mi familia por el apoyo que me han brindado a lo largo de todos los años de carrera universitaria y por siempre creer en mis capacidades, y particularmente a mi padre Jaime Alfaro Cabrera por ser un ejemplo de superación y perseverancia. Además, quiero agradecer a mi mejor amiga Mariana Venegas Ramirez y su familia, por ser una segunda familia para mi y brindarme apoyo y contención durante mi periodo de estudios.

Tabla de Contenido

1. Introducción a la robótica humanoide	1
1.1. Caso de estudio	2
1.2. Objetivos	3
1.2.1. Objetivo General	3
1.2.2. Objetivos Específicos	3
2. Marco teórico	5
2.1. Robot iCub	5
2.1.1. Caracterización del robot iCub a utilizar	5
2.1.2. iCub: Hardware	6
2.1.3. iCub: Software	6
2.2. YARP: Yet Another Robot Plataform	7
2.3. Algoritmos de Detección de Poses Humanas	9
2.3.1. Mediapipe Pose	9
2.4. Cinemática Directa e Inversa	10
2.4.1. Cinemática Directa	10
2.4.2. Cinemática Inversa	11
2.5. Estabilidad Estática y Dinámica	11
2.5.1. Conceptos relevantes	12
3. Estado del arte	13
3.1. Simultaneous Floating-Base Estimation of Human Kinematics and Joint Torques	13
3.2. Whole-Body Geometric Retargeting for Humanoid Robots	14
3.3. Model-Based Real-Time Motion Tracking Using Dynamical Inverse Kinematics	16
4. Diseño e implementación del sistema	18
4.1. Fundamentos de la solución: Human Dynamics Estimation	18
4.2. Diseño y estructura de la solución	19
4.2.1. Nuevo enfoque: sin sensores, sólo visión	19
4.2.2. Elementos y estructura del nuevo sistema	19
4.2.3. Alcance y limitaciones	20
4.3. Solución propuesta y subsistemas	20
4.3.1. Diagrama de bloques	20
4.3.2. Subsistema A: visión	21
4.3.3. Subsistema B: transformación de data	22

4.3.4.	Subsistema C: control	25
4.3.5.	Subsistema D: validación	29
4.3.6.	Sistema general	29
5.	Evaluación y análisis de resultados	30
5.1.	Método de evaluación por subsistema	30
5.1.1.	A: Visión	30
5.1.2.	B: Transformación de data	32
5.1.3.	C: Control	34
5.2.	Método de evaluación del sistema general	35
5.2.1.	Montaje experimental	35
5.2.2.	Parámetros de evaluación	35
5.2.3.	Teleoperación utilizando simulación	36
5.2.4.	Teleoperación utilizando el robot real - caso local	38
5.3.	Registro y análisis de resultados: subsistemas	42
5.3.1.	A: Visión	43
5.3.2.	B: Transformación de data	45
5.3.3.	C: Control	48
5.4.	Registro y análisis de resultados: sistema general	48
5.4.1.	Control en simulación	49
5.4.2.	Control en robot real	58
6.	Conclusiones	68
A.	Robot iCub: especificaciones técnicas	70
A.1.	Versiones.	70
A.2.	Rango de movimiento de articulaciones	71
	Bibliografía	71

Índice de Tablas

2.1. Grados de libertad del robot iCub v2.7	7
5.1. Precisión de detección de pose de BlazerPose para distintos conjuntos de datos. Valores declarados por Google [1].	30
5.2. Tiempos de inferencia para los modelos de detección de pose usados en MediaPipe Pose.	32
5.3. Tiempos de inferencia de Mediapipe Pose para 20 <i>frames</i> , realizando la inferencia dentro de un contenedor <i>Docker</i> y de forma local.	44
5.4. Tiempos de inferencia y envío de vector de datos para 20 <i>frames</i> (subsistema B).	47
5.5. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para las caderas.	51
5.6. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para las rodillas.	51
5.7. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para los talones.	52
5.8. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para los hombros.	52
5.9. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para los codos.	53
5.10. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para las caderas.	61
5.11. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para los talones.	61
5.12. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para los talones.	61
5.13. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para los hombros.	62
5.14. Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para los codos.	62
A.1. Versiones disponibles del iCub y sus principales características.	70
A.2. Rango de movimiento de las articulaciones para distintas aplicaciones (<i>Hardware</i> y <i>Software</i>).	71

Índice de Ilustraciones

1.1. Diagrama del sistema de teleoperación a desarrollar.	3
2.1. Robot iCub.	6
2.2. Puntos que detecta el modelo Mediapipe Pose.	10
3.1. Traje de sensores XSens utilizado para obtener información del movimiento del operador.	14
3.2. Diagrama del sistema propuesto por [2].	15
3.3. Resultados obtenidos utilizando reajuste de movimiento para diferentes robots humanoides.	15
3.4. Diagrama de sistema de control propuesto utilizando estimación dinámica.	16
3.5. Pruebas realizadas utilizando el robot iCub. Imagen recuperada del artículo [3].	17
4.1. Diagrama de bloques del sistema propuesto.	21
4.2. Puntos o <i>landmarks</i> seleccionadas para el sistema.	22
4.3. Dirección normal definida y rotación para obtener vector B a partir de A.	23
4.4. Modelamiento del cuerpo humano a base de <i>links</i> y <i>joints</i> en la pose T.	24
4.5. Diagrama de flujo del dispositivo <i>HumanStateProvider</i>	28
5.1. Montaje experimental para evaluación de estimación de ángulos.	33
5.2. Montaje experimental para experimento de comunicación mediante red YARP.	34
5.3. Montaje experimental para experimentos del sistema completo con resultados de simulación.	35
5.4. Montaje experimental para experimentos del sistema completo con resultados en el robot real.	35
5.5. Configuración de <i>Gazebo</i> para los experimentos en simulación.	37
5.6. Menú de articulaciones del robot a operar.	37
5.7. Interfaz gráfica del dispositivo <i>Control Board</i>	38
5.8. Inicio de <i>yarpserver</i> . A la izquierda, terminal antes de ejecutar el comando. A la derecha, terminal después de ejecutado el comando (se levanta el servidor <i>yarp</i>).	39
5.9. Inicio de <i>yarpmanager</i> . A la izquierda, terminal antes de ejecutar el comando. A la derecha, interfaz gráfica que se abre una vez ejecutado el comando anterior.	39
5.10. Conexión de nodos. Se seleccionan los nodos <i>icub-head</i> e <i>icubsrv</i> y se conectan utilizando el botón indicado por la flecha.	40
5.11. Conexión de nodos. Se seleccionan los nodos <i>icub-head</i> e <i>icubsrv</i> y se conectan utilizando el botón indicado por la flecha.	40
5.12. Inicio de <i>yarpmotorgui</i> para el robot iCub real.	41

5.13. Interfaz gráfica de control de los motores y actuadores presentes en las articulaciones del robot.	41
5.14. Variación de la longitud de las extremidades utilizando como inferencia <i>Pose Landmarks</i>	43
5.15. Variación de la longitud de las extremidades utilizando como inferencia <i>Pose World Landmarks</i>	43
5.16. Reconstrucción de pose detectada utilizando ángulos estimados por el subsistema B.	45
5.17. Envío y recepción correcta de paquetes de datos desde el subsistema B a un puerto de lectura YARP.	46
5.18. Conexión exitosa entre puerto de escritura creado usando <i>Python</i> y puerto de lectura definido usando comandos de <i>YARP</i>	47
5.19. Visualización (izquierda) y simulación (derecha) de los movimientos del robot iCub dada una pose objetivo.	48
5.20. Teleoperación de simulación de robot iCub v2.5 en tiempo real. El robot se encuentra fijo desde su cadera sin tener contacto con el suelo (superficie flotante).	49
5.21. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para la cadera izquierda.	53
5.22. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para la cadera derecha.	54
5.23. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para la rodilla izquierda.	54
5.24. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para la rodilla derecha.	55
5.25. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el tobillo izquierdo.	55
5.26. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el tobillo derecho.	56
5.27. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el hombro izquierdo.	56
5.28. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el hombro derecho.	57
5.29. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el codo izquierdo.	57
5.30. Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el codo derecho.	58
5.31. Teleoperación de robot iCub v2.7 en tiempo real. El robot se encuentra fijo desde su cadera sin tener contacto con el suelo (superficie flotante).	59
5.32. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para la cadera izquierda.	62
5.33. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para la cadera derecha.	63
5.34. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para la rodilla izquierda.	63
5.35. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para la rodilla derecha.	64
5.36. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el tobillo izquierdo.	64

5.37. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el tobillo derecho.	65
5.38. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el hombro izquierdo.	65
5.39. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el hombro derecho.	66
5.40. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el codo izquierdo.	66
5.41. Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el codo derecho.	67

Capítulo 1

Introducción a la robótica humanoide

En la era contemporánea, los robots humanoides que poseen un cierto grado de autonomía, han emergido de la convergencia entre la robótica y la inteligencia artificial. Un robot humanoide es una máquina de diseño antropomórfico creada para imitar la forma, función y comportamiento de un humano. Estos robots han surgido gracias a avances en mecánica precisa y el desarrollo de algoritmos sofisticados, dotándolos de la capacidad para llevar a cabo tareas que van desde la asistencia en la industria y la atención médica ¹, hasta la exploración espacial ² y la interacción social. Esta convergencia tecnológica ha desencadenado un cambio paradigmático en la forma en que los seres humanos interactúan con la maquinaria, planteando cuestiones intrigantes sobre la relación entre la tecnología y la humanidad en la sociedad contemporánea.

En términos de las implicaciones tecnológicas derivadas de este campo de estudio, el desarrollo de robots humanoides ha necesitado abarcar diversas áreas de investigación debido a que, para lograr una semejanza lo más precisa posible con el ser humano, estos robots deben no solo tener la capacidad de verse y moverse como un humano, sino también de replicar sus sentidos más relevantes mediante el uso de inteligencia artificial, tales como el tacto, la visión y el equilibrio (aunque se han desarrollado robots con capacidad para el gusto y el olfato, estos aspectos no resultan relevantes para las interacciones entre humano-robot). En particular, el desarrollo de la “visión” en los robots depende directamente de los avances en el campo de la *Visión por Computadora*.

La *Visión por Computadora* (en inglés *Computer Vision*) corresponde al campo de estudio que se centra en capacitar a las computadoras y máquinas para ver y comprender el mundo que las rodea. Ha resultado ser un factor determinante en diversas industrias, permitiendo que las máquinas logren percibir e interpretar información visual, conduciendo a avances en robótica e inteligencia artificial [4]. Algunos problemas relevantes de visión por computadora relacionados con la robótica incluyen el reconocimiento de objetos, el reconocimiento de humanos, el reconocimiento facial, el reconocimiento de gestos y la comprensión de escenas.

Paralelamente, el concepto de teleoperación en el contexto de la robótica humanoide se ha posicionado como una solución para superar los desafíos de distancia, seguridad y accesibilidad en

¹Robots in Healthcare, disponible en <https://www.ahu.edu/blog/robotics-in-healthcare>.

²Robot explorador en el planeta Marte: <https://mars.nasa.gov/mars-exploration/missions/mars2020/>.

la realización de tareas en donde se requiere la intervención o supervisión humana en entornos complejos o peligrosos. La teleoperación se refiere a la capacidad de controlar y dirigir a un robot desde una ubicación remota, utilizando interfaces tecnológicas que permiten a los operadores humanos extender sus habilidades a un robot. Este tipo de sistemas comúnmente poseen una estructura “operador-marioneta”, en donde existe un subsistema principal de operación que será utilizado por el operador, y un subsistema de control encargado de manejar de forma remota el robot (marioneta). Esta tecnología ha permitido a los humanos realizar tareas delicadas y complejas en entornos inaccesibles o peligrosos tales como exploración de lugares hostiles² y ejecución de tareas industriales de alto riesgo³. A medida que las capacidades de teleoperación evolucionan, su impacto en la industria, la investigación y la seguridad se vuelve cada vez más evidente, consolidando su posición como un componente esencial en el desarrollo los robots humanoides y su aplicación en la vida actual.

Cuando convergen los campos de la robótica humanoide, la visión computacional y la teleoperación, se logran sistemas complejos de robots capaces de percibir su entorno y proporcionar esta información al operador o sistema de control, permitiéndoles tomar decisiones ya sea de forma autónoma o bajo la dirección de un operador. Por ejemplo, algunos casos de uso de visión computacional en el sistema de control corresponden a reconocimiento de gestos y expresiones humanas en las interacciones “humano-robot”, permitiendo que el robot responda de manera apropiada y colabore de manera efectiva con humanos en diversas tareas, mientras que en los casos en que se utiliza para retroalimentación del operador tenemos ejemplos como la exploración de ambientes peligrosos, rescate, y teleoperación de robots industriales.

1.1. Caso de estudio

Los campos que abarca el problema a investigar son los de teleoperación, sistemas de control, robótica y visión computacional. En este caso de estudio se buscará teleoperar un robot humanoide en tiempo real mediante el diseño de un sistema “operador-marioneta”⁴, utilizando como referencia los movimientos de una persona captados con una cámara externa al robot.

Una de las características que diferenciará a este caso de estudio de otros presentes en la literatura, es que el control del cuerpo del robot se hará utilizando exclusivamente información extraída de imágenes en tiempo real, captadas con una cámara fija, sin utilizar ningún otro tipo de sensores. Además, el sistema a desarrollar será construido de tal forma que la teleoperación se pueda realizar de forma remota o local (ver Figura 1.1), eliminando la barrera de distancia para el control del robot e incluso abriendo paso a nuevos trabajos futuros que busquen estudiar el uso de robots humanoides como avatares, donde el usuario pueda recibir retroalimentación del entorno desde la perspectiva del robot. En cuanto al control del robot, se respetarán las restricciones físicas y mecánicas de este, particularmente las limitaciones cinemáticas del robot, procurando su estabilidad durante la ejecución de los movimientos y poses requeridas por el operador.

En el presente documento se registra la investigación, desarrollo, experimentos y conclusiones del estudio del tema “*Teleoperación de Robot iCub en tiempo real*”. En el Capítulo 1.2 se defi-

³Artículo sobre robótica colaborativa disponible en <https://www.atriainnovation.com/la-robotica-colaborativa-en-entornos-peligrosos>

⁴El nombre hace alusión a que el operador en este caso controlará completamente los movimientos del robot, como si operara una marioneta.

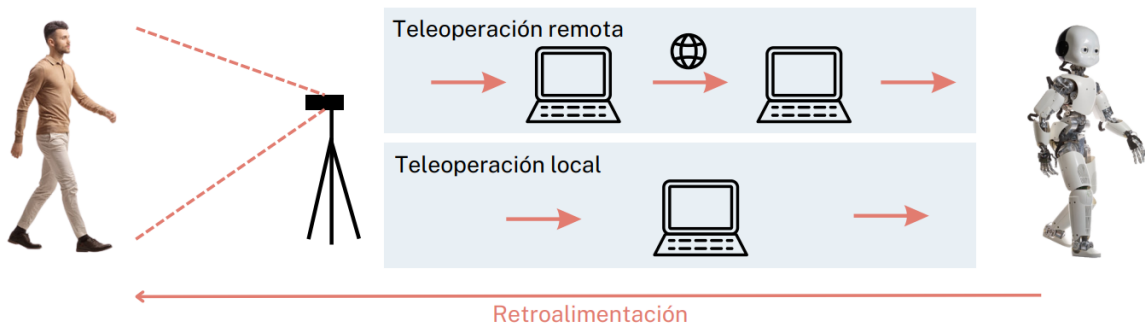


Figura 1.1: Diagrama del sistema de teleoperación a desarrollar.

ne el objetivo general y los objetivos específicos de este trabajo; en el Capítulo 2 se realiza una investigación de los conceptos claves para la comprensión y resolución del problema abordado. Posteriormente, en el Capítulo 3 se incluye literatura relevante para el desarrollo de esta memoria e investigaciones con un enfoque similar a la propuesta, mientras que en el Capítulo 4 se diseña e implementa una solución para el problema en cuestión. En el Capítulo 5 se desarrollan experimentos para la evaluación del sistema implementado, indicando y analizando los resultados esperados y obtenidos de este proyecto y, finalmente, en el Capítulo 6 se analiza la efectividad de la solución propuesta.

1.2. Objetivos

1.2.1. Objetivo General

El objetivo principal de este trabajo es lograr teleoperar en tiempo real un robot humanoide, en particular un robot iCub version 2.7, a partir de información visual capturada utilizando solo una cámara de video externa al robot.

1.2.2. Objetivos Específicos

Para lograr el objetivo principal, el trabajo se centrará en los objetivos específicos de:

- Desarrollar un sistema que permita extraer información relevante de posición y velocidad de las articulaciones y/o extremidades de un humano utilizando algoritmos de visión computacional.
- Desarrollar un sistema de comunicación que conecte los subsistemas de visión y control del robot a través de la transformación de datos entregados por el sistema de visión, para compatibilizarlos con el sistema de control.
- Desarrollar un sistema de control que permita operar los movimientos del robot iCub a partir de la transformación de los datos entregados por el sistema de visión computacional que caracteriza las poses humanas.
- Validar el funcionamiento del sistema creado en simulaciones y en el robot real.
- Evaluar el desempeño del sistema de forma cualitativa y cuantitativa.

En el siguiente capítulo se realiza un estudio de conceptos previos que tendrán gran relevancia

en el desarrollo de esta investigación.

Capítulo 2

Marco teórico

En el presente capítulo se estudiarán las características principales del robot humanoide a utilizar junto a el *software* requerido para la utilización y programación de este. Se estudiarán conceptos relacionados al campo de visión computacional, particularmente de detección de poses humanas, y finalmente se estudiarán conceptos relevantes de dinámica, cinemática y estabilidad relacionado al de robots.

2.1. Robot iCub

iCub es un robot humanoide desarrollado en el año 2004 por el *Consortio RobotCub* y construido en 2010 por el Instituto Italiano de Tecnología (IIT)[5] . Se caracteriza por tener el tamaño y la apariencia de un niño: mide alrededor de 105 centímetros, pesa 33 kilogramos y posee una estructura antropomórfica (Figura 2.1). Fue construido principalmente para su uso en investigación en áreas como la interacción humano-robot, el desarrollo de habilidades motoras, la percepción y el aprendizaje, aunque también se ha utilizado como herramienta para desarrollar y probar algoritmos y modelos teóricos relacionados con la cognición humana, la inteligencia artificial e incluso como robot de servicio.

Gracias a sus extremidades y articulaciones, el iCub, en sus versiones más recientes, puede realizar movimientos tales como caminar, gatear y mantenerse erguido; y utilizando sus manos, que poseen tres dedos de movimiento independiente (pulgar, índice y dedo medio) y dos de movimiento conjunto para mayor soporte (anular y pulgar), puede completar tareas sofisticadas de manipulación de objetos. Los grados de libertad de sus extremidades (brazos, manos, piernas y pies), al igual que las especificaciones de la cabeza y de los sensores de sonido y táctiles varían dependiendo de la versión del robot. Los detalles de las diferentes versiones del robot se encuentran en el Anexo A.1.

2.1.1. Caracterización del robot iCub a utilizar

El robot con el que se validará el sistema de imitación de poses en tiempo real corresponde al *iCub versión 2.7*. Este cuenta con 53 actuadores (que otorgan los 53 grados de libertad) ubicados a lo largo del cuerpo y que le permiten mover brazos, manos, piernas, pies, cabeza y torso. En cuanto a sensores, posee cámaras estéreo en los ojos, micrófonos, sensores de fuerza/toque, giroscopios y



Figura 2.1: Robot iCub.

acelerómetros, sensores táctiles que emulan la piel y *enconders* en todos sus articulaciones o *joints*. En el rostro cuenta con un panel LED que mediante su configuración permite representar la boca y las cejas, generando expresiones faciales. El robot se compone de 6 sub-sistemas: *cabeza*, *brazo izquierdo*, *brazo derecho*, *torso*, *pierna izquierda* y *pierna derecha*.

2.1.2. iCub: Hardware

El iCub cuenta con actuadores asociados a sus subsistemas, los que definen el rango de movimientos de estos. En la Tabla 2.1 se detallan los grados de libertad presentes en cada subsistema y/o componente del iCub.

Cada actuador tiene un rango de movimiento limitado, que define los límites de cada articulación. En el caso del iCub, este se conforma únicamente de *articulaciones rotacionales*, por lo que el rango de movimiento corresponderá al desplazamiento angular máximo de la articulación. En la Tabla A.2 del Anexo A.2 se encuentran definidos los límites para cada actuador.

Para poder funcionar, el robot se alimenta a través de un cable de energía conectado en la espalda o “mochila” del robot. El otro extremo del cable de energía se conecta a una fuente configurada a *40V* y *2A*. Para la comunicación con un computador, se utiliza un cable *Ethernet* conectado también a la espalda o “mochila” del robot. En el costado de esta misma “mochila” se encuentran los botones de encendido/apagado de los motores y la CPU del robot.

2.1.3. iCub: Software

Para el control del robot, el IIT provee un software de código abierto escrito en su mayoría en C++. *Robotology-Superbuild*¹ es el repositorio de *GitHub* que permite automáticamente descargar y compilar todo el software utilizado para operar el iCub. Este cuenta con simuladores, módulos e interfaces de control de los motores del iCub e integración con YARP. En la siguiente sección se describirá YARP.

¹Disponible en <https://github.com/robotology/robotology-superbuild>.

Tabla 2.1: Grados de libertad del robot iCub v2.7

Componente	# de grados de libertad	Notas
Ojos	3	Vergencia independiente, inclinación común
Cabeza	3	Presentes en el cuello, permiten los movimientos de <i>roll</i> , <i>pitch</i> y <i>yaw</i>
Torso	3	Permiten los movimientos de <i>roll</i> , <i>pitch</i> y <i>yaw</i>
Brazos	14 (7 cada uno)	3 grados de libertad presentes en el hombro, 1 en el codo y 3 en la muñeca
Manos	18 (9 cada una)	Dedos pulgar, índice y medio con movimientos independientes, dedos anular y pulgar presentan movimiento común. El pulgar puede rotar sobre la palma
Piernas	12 (6 cada una)	6 grados de libertad permiten caminar

2.2. YARP: Yet Another Robot Platform

YARP (*Yet Another Robot Platform*) es un conjunto de bibliotecas, protocolos y herramientas *framework* de código abierto escrito en C++ creado para el desarrollo de software en sistemas robóticos e investigación [6] [7]. Fue desarrollado por el laboratorio de robótica avanzada de la Universidad de Génova en Italia y permite una comunicación y control eficientes de subsistemas robóticos. Gracias a su arquitectura modular y desacoplada, los productos desarrollados con YARP son fácilmente escalables, permitiendo la adaptación del software desarrollado según los requisitos específicos de cada robot. YARP no corresponde a un sistema operativo, sino que es un paquete con múltiples herramientas que ayudan a definir y organizar la comunicación entre sensores, procesadores, actuadores y otros dispositivos permitiéndoles estar débilmente acoplados (o totalmente desacoplados)², lo que permite realizar cambios en un componente sin afectar en gran medida al resto del sistema, favoreciendo la evolución gradual de estos.

Los componentes de YARP se pueden separar en 3 categorías: *YARP_os*, *YARP_sig* y *YARP_dev*. *YARP_os* se encarga de la interfaz y sistema(s) operativo(s) para brindar una transmisión de datos simple entre varios hilos presentes en varias máquinas. YARP está diseñado para ser independiente del sistema operativo, permitiendo su uso en *Linux*, *Microsoft Windows*, *Apple macOS* e *iOS*, *Solaris* y *Android*. *YARP_sig* ejecuta tareas comunes de procesamiento de señales (visuales, sonoras), admitiendo su interconexión con bibliotecas de uso común tales como OpenCV. *YARP_dev* corresponde a la interfaz que contiene los dispositivos comunes utilizados en el campo de la robótica: cámaras digitales, paneles de control de motores, capturadores de imágenes, sensores, entre otros³.

Esta librería soporta la transmisión de datos a través de diferentes protocolos de bajo nivel tales como: *TCP*, *UDP*, *MCAST* (multi-cast) y *shared memory*, los que son conocidos como *carriers*.

²Más información en <https://www.yarp.it/latest/>.

³Para más información sobre los módulos de YARP ingresar a https://yarp.it/latest/what_is_yarp.html.

El modelo de comunicación de YARP es *transporte-neutral*, por lo que los datos a transmitir estarán desacoplados de los detalles de la red de comunicación y protocolos en uso, permitiendo que pueda estar en uso más de un protocolo en forma simultánea. El sistema de comunicación entre los distintos módulos en YARP, generalmente, sigue el patrón de diseño *Observer*, el cual notifica a múltiples objetos “observadores” sobre los eventos que acontecen en el objeto “observado”. El sistema de comunicación en YARP se da entre la conexión de objetos llamados puertos o *ports*. El conjunto de puertos y conexiones entre ellos conforman la red o *YARP network*.

Para comprender el funcionamiento de este *framework*, es necesario conocer la siguiente terminología:

- **YARP Ports**

Un puerto YARP (o *YARP Port* en inglés) es el canal que permite la comunicación e intercambio de data entre los diferentes módulos de un software. Mediante estos *ports* se envían y reciben mensajes, comandos y datos entre los componentes, dispositivos o subsistemas de un sistema robótico.

Cada puerto YARP tiene un identificador único asociado. Este identificador corresponde al nombre con el que se registra el puerto en el servidor. El sistema de comunicación está construido de tal forma que, solo conociendo el nombre del puerto es posible comunicarse con este desde cualquier dispositivo. Además, puede establecer conexiones con cualquier cantidad de puertos para entregar mensajes a cualquier cantidad de observadores, en cualquier cantidad de procesos, distribuidos en cualquier cantidad de máquinas, utilizando una variedad de protocolos de comunicación.⁴

- **YARP Devices**

Los dispositivos YARP (o *YARP devices* en inglés) permiten la comunicación entre módulos de *software* y dispositivos de *hardware* tales como sensores, actuadores, cámaras, entre otros. Un *YARP device* corresponde a un módulo definido para actuar a modo de interfaz de comunicación y transferencia de datos entre los componentes físicos del robot y el software de control. De esta forma, proporcionan una capa de abstracción que facilita la programación del robot, independientemente de su configuración específica de *hardware*, lo cual permite la escalabilidad del sistema de control.

Para cada componente físico particular o familia de componentes se definirá un *YARP device* que permita integrar este dispositivo al sistema, considerando la lógica necesaria para establecer la comunicación con el *hardware* correspondiente y ofreciendo una interfaz uniforme y consistente al tipo de datos y comandos del aparato físico. Con esto, el desarrollo de *software* simplemente procura ser congruente con las clases de dispositivos definidos en YARP, y no debe adaptarse a todas las posibles variantes de *hardware* que puedan existir para un mismo tipo de componente. Por ejemplo, si se desea integrar cámaras al sistema, el *software* de control se desarrollará para comunicarse con un controlador de cámara definida (o dispositivo YARP clase cámara) y no con el dispositivo de cámara físico de forma directa.

⁴Para más información referirse a <https://yarp.it/latest/>.

2.3. Algoritmos de Detección de Poses Humanas

El mapeo y detección de poses humanas es un problema de visión computacional que consiste en identificar y comprender la posición de un cuerpo humano a partir del procesamiento de imágenes o videos. Este problema implica identificar y localizar diferentes articulaciones y/o partes del cuerpo humano claves para poder replicar o comprender la pose presente en la imagen analizada, y ser capaz de generar a partir de estas un set de datos que permita describir y reproducir la pose detectada.

Antes de comenzar a estudiar los algoritmos de detección de pose humana, es importante conocer los conceptos de *Top-Down* y *Bottom-Up*. Los modelos de detección de pose de tipo *Top-Down* implementan primero un detector de personas, y en base a la región en la que se estima que estará ubicada la persona, realizan una estimación de la posición de las extremidades y finalmente la pose. Los modelos de tipo *Bottom-Up* realizan el procedimiento inverso: primero estiman la pose de la persona, y luego estiman la región que ocupa la persona en la imagen.

Además, para facilitar la comprensión de este trabajo, se define la siguiente terminología:

- **Joint:** hace referencia a las articulaciones de un cuerpo.
- **Link:** enlace o vector que representa una extremidad. Dos *links* estarán unidos por un *joint*.
- **Configuración padre-hijo:** hace referencia a la relación entre dos *links* unidos por un *joint*. El *link* padre corresponde al que se encuentra más cerca de la base definida del robot, mientras que el *link* hijo corresponde al más lejano de la base del robot. El *link* hijo siempre tendrá un grado de libertad más que el *link* padre
- **Grados de libertad:** cantidad de movimientos independientes que un robot o sistema mecánico puede realizar.

Para identificar la pose objetivo que se desea que el robot alcance, es necesario contar con un algoritmo de detección de poses humanas. Existen en la actualidad modelos entrenados y de código abierto para la detección de objetos y estimación de poses humanas que poseen un muy buen desempeño. A continuación se describe brevemente uno de estos algoritmos, correspondiente a *MediaPipe Pose*.

2.3.1. MediaPipe Pose

*MediaPipe Pose*⁵ corresponde a una de las soluciones de *Machine Learning* de código abierto de *MediaPipe* desarrollada por *Google*. Consiste en un *framework* de visión computacional que permite la detección y rastreo de poses humanas capaz de inferir la posición de un cuerpo en un espacio tridimensional a partir de una imagen o video. Mediante la detección e inferencia de la ubicación de 33 *joints* (ver Figura 2.2) del cuerpo humano correspondiente a zonas como hombros, codos, rodillas, caderas y otros, logra trazar un diagrama representativo de un cuerpo humano con la pose detectada. Para cada punto detectado, *MediaPipe Pose* genera 4 valores: posición del punto en x , y , z y un cuarto valor llamado “visibilidad”, que otorga un valor de 1 a 100 indicando la visibilidad del punto estimado en la imagen o video.

⁵Disponible en https://developers.google.com/mediapipe/solutions/vision/pose_landmarker.

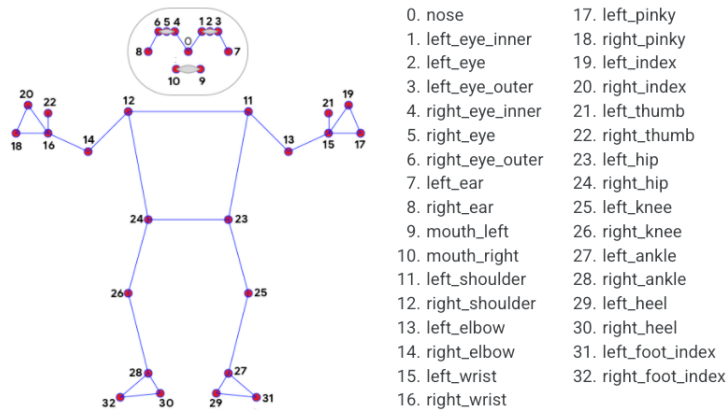


Figura 2.2: Puntos que detecta el modelo Mediapipe Pose. Cada número de punto indica una articulación distinta en el cuerpo humano.

Basado en *Machine Learning*, *Mediapipe Pose* cuenta además con segmentación de cuerpo del fondo, permitiendo hacer seguimiento del desplazamiento del humano en el video procesado. Esta solución implementa una arquitectura de modelo detector-rastreador (*Top-Down*) de dos etapas: iniciando con el detector, primero se localiza la región de interés en donde se encuentra la persona/postura dentro del marco. Posteriormente, el rastreador predice los puntos de referencia de la pose y, se determina la máscara de segmentación dentro de la región de interés, usando el marco recortado de la región de interés como entrada.

Debido a la estructura de esta solución, en los casos donde se analiza video el detector se invoca solo según sea necesario, es decir, para el primer fotograma y cuando el rastreador ya no puede identificar la presencia de la pose del cuerpo en el cuadro anterior. Para los fotogramas siguientes, el modelo simplemente deriva la región de interés a partir de los puntos de la pose detectada en el fotograma anterior.

2.4. Cinemática Directa e Inversa

La cinemática es el estudio del movimiento de objetos. Esta no considera las fuerzas que actúan en los objetos y que, por consiguiente, producen su movimiento, sino que se enfoca en describir matemática y geoméricamente las trayectorias, velocidades, aceleraciones y posiciones de los objetos en movimiento.

En robótica, la cinemática estudia el movimiento de los conjuntos de sólidos rígidos (o *links*) conectados por articulaciones (*joints*) que conforman las partes de un robot. Para poder desarrollar algoritmos de control para un robot, resulta necesario conocer los conceptos de *cinemática directa* y *cinemática inversa*.

2.4.1. Cinemática Directa

La *cinemática directa* busca encontrar la posición y orientación en el espacio del extremo final de alguna cadena cinemática, conociendo los valores de las coordenadas articulares (en inglés *joint coordinates*) y las características geométricas del robot. El problema cinemático directo consiste en encontrar una matriz de transformación que relacione el sistema de coordenadas ligado al cuerpo en

movimiento (coordenadas articulares) con respecto al sistema de coordenadas 3D que se utilizará como referencia.

La matriz de transformación homogénea a encontrar, corresponde a una matriz de 4×4 que posee la estructura mostrada en la Ecuación (2.1), donde M_{rot} corresponde a la matriz de rotación, V_{pos} al vector de posición, y donde los vectores \vec{n} , \vec{s} y \vec{a} son vectores ortogonales unitarios y $V_{\text{pos}} = \vec{p}$ es el vector que describe la posición (x, y, z) del origen del sistema actual, respecto del sistema de referencia.

$$T = \begin{bmatrix} M_{\text{rot}} & V_{\text{pos}} \\ 0_{1 \times 3} & 1 \end{bmatrix} = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \vec{n} & \vec{s} & \vec{a} & \vec{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

2.4.2. Cinemática Inversa

La *cinemática inversa*, tal como sugiere su nombre, corresponde a realizar el proceso inverso a la cinemática directa. En este se busca encontrar las variables articulares $\vec{q} = [q_1, q_2, \dots, q_n]^T$ a partir de una posición y orientación conocida para el extremo de una cadena cinemática. El procedimiento de obtención de las ecuaciones para determinar \vec{q} dependerá de la configuración del robot, y la solución puede no ser única, existiendo la posibilidad de que k vectores \vec{q} posicionen y orienten la cadena cinemática de la misma forma.

Existen distintos tipos de solución al problema de cinemática inversa:

- **Métodos geométricos:** se utilizan para posicionar variables articulares en robots de pocos grados de libertad o para solo los primeros grados de libertad. Utilizan relaciones geométricas y trigonometría.
- **Matrices de transformación homogénea:** despeja las n variables del vector \vec{q} en función de las componentes de los vectores \vec{n} , \vec{s} , \vec{a} , \vec{p} , conociendo previamente el modelo de cinemática directa del robot.
- **Desacoplamiento cinemático:** realiza una separación entre orientación y posicionamiento.
- **Soluciones numéricas (iterativas):** no se garantiza que puedan ser empleados en tiempo real.

2.5. Estabilidad Estática y Dinámica

En el campo de la robótica, la estabilidad de un robot consiste en su capacidad de mantener el balance. Para que un robot sea estable, la proyección de su centro de gravedad debe ser contenido en el polígono de soporte que tiene como vértices los puntos de contacto del robot con su superficie de apoyo. Mientras más puntos de apoyo tenga el robot, mayor será el área del polígono y, por lo tanto, será más estable. Existen dos tipos de estabilidad: estabilidad estática y estabilidad dinámica.

Un robot (en general, un mecanismo) se dice *estáticamente estable* cuando estando todas sus articulaciones fijas, es capaz de mantener el balance. La *estabilidad dinámica*, por otra parte, hace referencia a la capacidad de un robot de, a través del movimiento de sus articulaciones, volver a alcanzar la estabilidad.

La locomoción de un robot bípedo, como el iCub, ha sido un desafío para la comunidad científica por muchos años. Los conceptos de estabilidad estática y dinámica, en este contexto, son cruciales, pues mantener el balance de un robot bípedo en movimiento es necesario para realizar tareas como caminar, correr, saltar y, en el contexto del presente proyecto, imitar poses humanas.

En problemas de control de robot bípedos, como el problema de caminar, existen secuencialmente situaciones en las que el robot se encuentra con uno o dos puntos de apoyo en el piso, de este modo, oscilando entre estabilidad estática (y una cadena cinemática cerrada) e inestabilidad estática (y una cadena cinemática abierta) [8]. Adicionalmente, el/los punto(s) de contacto entre el piso y el/los punto(s) de apoyo del robot pueden considerarse como articulaciones pasivas, las cuales no son directamente controlables.

2.5.1. Conceptos relevantes

Un concepto especialmente relevante asociado al balance de un robot bípedo es el “punto de momento cero” (en inglés *zero moment point* [ZMP]) [8]. El ZMP es un punto en el suelo (más precisamente, en el polígono de soporte del robot), en el que las fuerzas inerciales y gravitacionales tienen un efecto neto nulo en componentes horizontales (ejes X e Y , considerando al eje Z como ortogonal a la superficie). Cuando el ZMP se encuentra dentro del polígono de soporte del robot, entonces el robot es dinámicamente estable. Cuando el ZMP se acerca a algún límite de la región de soporte, esta es una indicación de que el robot va a caer (e.g., si el polígono de soporte corresponde a la suela del pie del robot, va a comenzar a rotar en torno al borde de dicho pie, siendo este borde al que el ZMP se acercó). Un ZMP fuera del polígono de soporte es denotado como *punto de momento cero ficticio* o FZMP (en inglés “fictitious zero moment point”), y su relación con el ZMP es que su distancia a él modulará la intensidad con la que la rotación en torno al borde del polígono de soporte se desarrollará [8].

Otro concepto relevante, que tiene una conexión con el ZMP (además del polígono de soporte), corresponde al “centro de presión” (en inglés *center of pressure* [CoP]). La presión entre el contacto del robot y el piso puede ser representada como una fuerza actuando en el CoP. De este modo, cuando el robot es dinámicamente estable, el ZMP y el CoP coinciden, pero si el robot deja de ser estable, y solo existe el FZMP, entonces el CoP no podría coincidir con el ZMP (en este caso la fuerza en el CoP no cumple las condiciones del ZMP) [8].

Finalmente, considerando la complejidad de los robots humanoides (en general), estos tienden a ser modelados a través de simplificaciones con el fin de resolver de forma práctica algunos de los sub-problemas asociadas a la caminata bípeda. Una simplificación bastante popular consiste en modelar la dinámica del robot a través de las ecuaciones de movimiento de una masa puntual que representa su centro de masa (en inglés *center of mass* [CoM]), empleando diversas variantes (e.g. [9]).

Capítulo 3

Estado del arte

A continuación, se realiza una revisión bibliográfica de artículos de investigación de relevancia para este proyecto. En particular, se revisan 3 artículos que presentan soluciones al problema de control y operación de un robot utilizando como referencia los movimientos de un sujeto humano bajo la interacción operador-avatar.

3.1. Simultaneous Floating-Base Estimation of Human Kinematics and Joint Torques

En *Simultaneous Floating-Base Estimation of Human Kinematics and Joint Torques* [10] se presenta una metodología estocástica para la estimación simultánea de la cinemática y dinámica de todo el cuerpo humano, considerando la pelvis como base flotante (el término *base flotante* se refiere a la consideración de que parte del cuerpo humano no está en contacto con una superficie fija). La estimación es calculada utilizando una combinación de sensores (ver Figura 3.1) capaz de proporcionar valores estimados de la cinemática y dinámica del total del cuerpo humano (torques articulares, fuerzas internas sobre las articulaciones y fuerzas externas actuando sobre los segmentos), en donde la posición y velocidad de la base flotante (en este caso, la pelvis) no es asumida como conocida.

En este caso de estudio se considera la dinámica del el cuerpo humano como un sistema en conjunto (con la pelvis como base flotante) y desarrollan un algoritmo para detectar el contacto de los pies humanos con el suelo a través de sensores de fuerza y, así, lograr identificar si el usuario humano está apoyando uno o dos pies en la tierra. La solución propuesta cuenta con 4 objetivos: generar un Formato de Descripción Universal de Robot (Universal Robot Description Format o URDF por sus siglas) con el cual configurar las características físicas del sujeto, desarrollar algoritmos para la estimación de la configuración cinemática del humano, implementar un algoritmo de reconocimiento del estado de apoyo con los pies y estimar simultáneamente la cinemática y dinámica considerando un modelo con una base flotante.

Para comprobar el funcionamiento de la propuesta, se llevó a cabo una sesión de prueba con un sujeto masculino saludable equipado con un traje con 17 sensores de movimiento (*Xsens*) para capturar información cinemática del cuerpo completo y un par de zapatos con sensores para la

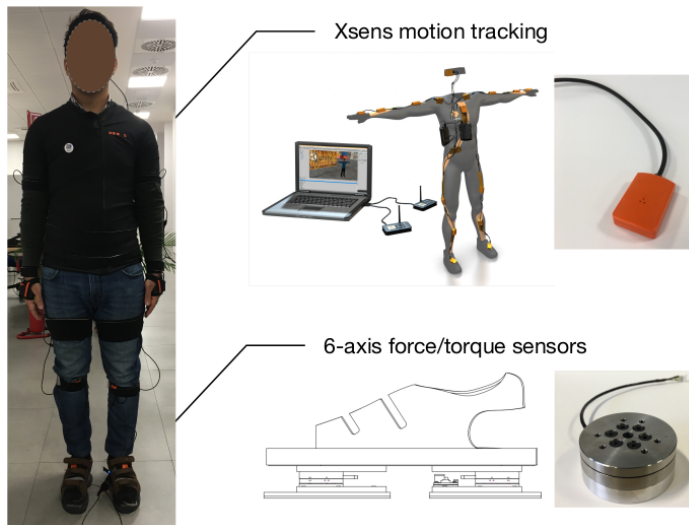


Figura 3.1: Traje de sensores XSens utilizado para obtener información del movimiento del operador. Imagen recuperada del artículo [10].

detección de fuerzas de reacción del suelo. Los resultados obtenidos muestran que los valores estimados que describen la cinemática del cuerpo humano concuerdan con los valores medidos utilizando los sensores. Con esto, se valida la solución propuesta.

El repositorio de *GitHub human-dynamics-estimation*¹ contiene todo el código y los dispositivos desarrollados dentro de este artículo para la estimación de cinemática y dinámica del cuerpo humano, escrito en C++.

3.2. Whole-Body Geometric Retargeting for Humanoid Robots

En este artículo estudiado [2] se presenta un *framework* para la teleoperación de robots humanoides utilizando un nuevo enfoque para el reajuste del movimiento (*motion retargeting*), a través de la cinemática inversa sobre el modelo del robot. Proponen un método de reajuste de movimiento escalable, permitiendo la teleoperación de diferentes modelos de robots por diferentes usuarios humanos realizando cambios mínimos en el sistema propuesto. Utilizando modelos antropomórficos del cuerpo humano que son independientes de las dimensiones del usuario, logran obtener las referencias necesarias para el control del robot.

El problema que este trabajo busca resolver es que los sistemas de control propuestos en la literatura carecen de la habilidad de adaptar fácilmente los sistemas de control a diferentes robots y usuarios con diferentes geometrías, dinámicas y cinemáticas. Además, uno de los problemas considerandos en esta investigación es que el control del robot debe ser tal que este no pierda su equilibrio ni caiga, pero manteniendo su capacidad de maniobreo y habilidad de manipulación.

Respecto al equipamiento utilizado en la solución propuesta para la teleoperación de cuerpo completo, se cuenta con los siguientes dispositivos:

- Unas gafas de realidad virtual, en las cuales se transmitirá el campo de visión del robot y que

¹Repositorio disponible en <https://github.com/robotology/human-dynamics-estimation>.

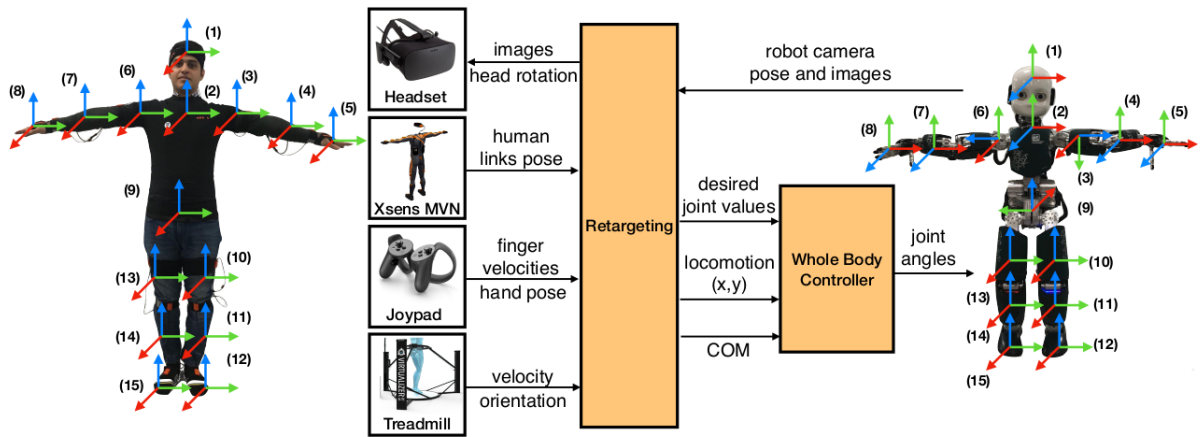


Figura 3.2: Diagrama del sistema propuesto por [2]. Imagen recuperada del artículo [2].

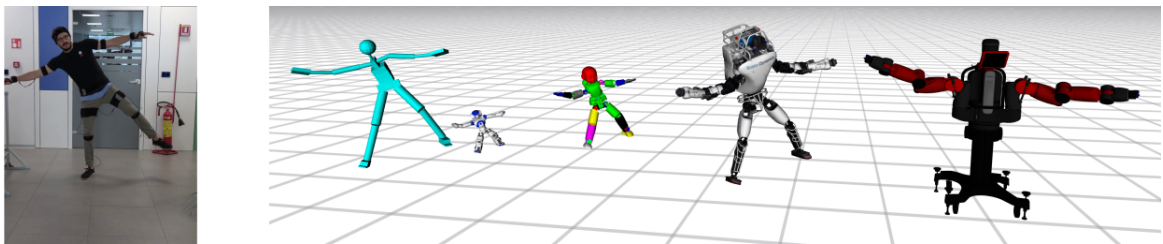


Figura 3.3: Resultados obtenidos utilizando reajuste de movimiento para diferentes robots humanoides. Imagen recuperada del artículo [2].

servirán para que el usuario reciba retroalimentación visual.

- *Joypads*, que se utilizarán para controlar las manos del robot.
- Una plataforma de caminata virtual (*Cyberith Virtualizer VR Treadmill*), con la que se obtendrán las velocidades lineal y angular como información de la locomoción del usuario.
- Un traje de cuerpo completo de sensores (*Xsens Suit*) que permitirá obtener la información cinemática de las extremidades del cuerpo del usuario.

Estos cuatro dispositivos generaran los valores de entrada al sistema de control propuesto (ver Figura 3.2). Con esto, logran mapear geoméricamente los movimientos del usuario a un modelo cinemático humano, y posteriormente mapear los movimientos de este modelo al modelo del robot que se desea teleoperar. Para el control del modelo cinemático humano, plantean el problema de reajuste de movimiento como uno de cinemática inversa, dada solo la rotación y velocidad angular de las extremidades del modelo humano y del modelo robótico. Debido a esto, los cambios en el usuario humano (y sus dimensiones) o en la geometría del robot a operar no afectan el sistema ya que, identificando la rotación y velocidad relativa del humano y ajustando las respectivas partes del robot logran obtener la posición y movimiento deseado del robot independiente de la geometría de este. En la Figura 3.3 se muestran los resultados obtenidos para la teleoperación de diferentes tipos de robots humanoides a partir de un único operador.

El trabajo expuesto en este artículo ayudará a fundar las bases de este trabajo de memoria, y aportará en gran medida como antecedente para el desarrollo de la investigación.

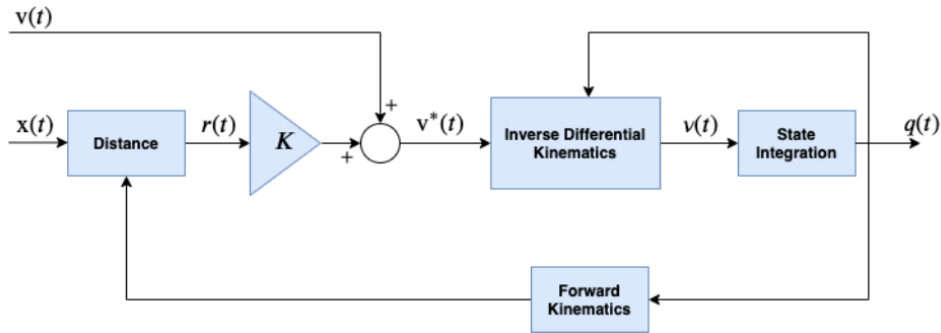


Figura 3.4: Diagrama de sistema de control propuesto utilizando estimación dinámica. Imagen recuperada del artículo [3].

3.3. Model-Based Real-Time Motion Tracking Using Dynamical Inverse Kinematics

En la investigación realizada en [3] se presenta un esquema para el seguimiento de movimiento en tiempo real de modelos humanos o humanoides altamente articulados. Este esquema busca optimizar los algoritmos de seguimiento para aplicaciones demandantes y críticas de tiempo. El método propuesto consiste en la aplicación de estrategias basadas en cinemática inversa dinámica a modelos de base flotante para optimizar su tiempo de cómputo.

Una de las soluciones más comunes para la resolución en tiempo real del problema de cinemática inversa para modelos altamente articulados es la *optimización instantánea*, que formula este problema como uno de optimización no lineal y lo resuelve numéricamente mediante la aplicación de algoritmos iterativos. La optimización instantánea presenta un buen desempeño y tiempo de convergencia en aplicaciones comunes de robótica, pero su velocidad de convergencia no es suficiente en situaciones más complejas como la inferencia crítica de tiempo en modelos altamente articulados como lo es el modelo humano. La solución investigada en este trabajo es la aplicación de la *optimización dinámica*, que replantea el problema de cinemática inversa como un problema de control, y que lograría disminuir el tiempo de cómputo ya que solo sería necesaria una iteración para lograr la convergencia de la solución.

Con el método de optimización dinámica se busca obtener los valores de posición, orientación, velocidad lineal y velocidad angular para alcanzar la configuración objetivo. Esto se puede plantear como minimizar la distancia entre los valores del modelo y los medidos. La solución de optimización dinámica es modelada como se muestra en la Figura 3.4. En esta se identifican 3 partes principales: la corrección de las velocidades objetivos a partir de la retroalimentación residual, la inversión del modelo diferencial cinemático para obtener las velocidades objetivo, y la integración numérica de las velocidades objetivo para obtener la posición y orientación objetivo. El diagrama de la solución se muestran en la Figura 3.4.

Para probar la solución propuesta se realizan experimentos utilizando como método de entrada la data generada por un humano utilizando el traje de sensores *XSens Awinda*, captando pose y velocidad para 23 extremidades humanas. Los modelos objetivos (es decir, los modelos que se desean controlar con los valores de entrada) corresponden a dos modelos con 23 extremidades, con 66 y 48 grados de libertad, y además se realizan pruebas en un modelo humanoide del robot



Figura 3.5: Pruebas realizadas utilizando el robot iCub. Imagen recuperada del artículo [3].

iCub (Figura 3.5), compuesto por 15 extremidades físicas y 34 grados de libertad. Se realizaron 3 actividades objetivo: pose T donde el sujeto se mantendrá con ambos pies en el suelo, caminata en una trotadora a una velocidad de 4 km/h, y corrida en una trotadora a 10 km/h.

Los resultados obtenidos indican que los errores producidos utilizando optimización dinámica son comparables a los producidos utilizando optimización instantánea, a excepción del caso del modelo humano con 66 grados de libertad en la actividad de correr, en donde el cálculo de la orientación presenta una precisión más baja. En el caso de la actividad de pose T, el cálculo de la velocidad angular obtuvo una precisión más alta con optimización dinámica. En cuanto a tiempos de cómputo, con la optimización dinámica se logra reducir un promedio de 3 ms. Los experimentos con el iCub real lograron determinar que para este es posible seguir los movimientos humanos con un tiempo de desfase menor a 300 ms. Este trabajo demuestra que es posible la teleoperación del robot iCub, con un tiempo de desfase mínimo.

Capítulo 4

Diseño e implementación del sistema

Considerando el estudio y revisión bibliográfica realizada, se procede a diseñar y desarrollar la solución propuesta.

4.1. Fundamentos de la solución: Human Dynamics Estimation

La solución a desarrollar se basa en la investigación realizada en el artículo *Whole Body Geometric Retargeting for Humanoid Robots, Simultaneous Floating-Base Estimation of Human Kinematics and Joint Torques* y el repositorio de GitHub *human-dynamics-estimation*¹. Este repositorio cuenta con una colección de dispositivos, módulos, interfaces y aplicaciones que permiten estimar en tiempo real la cinemática y dinámica de un humano a través del mapeo de información cinemática de este, obtenida utilizando sensores de movimiento adheridos al cuerpo, y realizar la transformación de la pose para que pueda ser replicada por un robot.

La principal característica de este repositorio es que resuelve el problema de cinemática inversa en tiempo real para replicar la configuración deseada en un modelo *URDF*² humano modelado utilizando extremidades compuestas de sólidos rígidos. Luego, si se desea replicar la pose utilizando un robot humanoide en particular, el controlador utilizará como referencia la pose del modelo humano para el *retargeting* de la pose, evitando así tener que resolver nuevamente el problema de cinemática inversa. Esta característica resulta beneficiosa ya que permite que el sistema sea escalable a diferentes tipos de robots humanoides: para extrapolar la posición a diferentes modelos de robots sólo se necesita configurar las secciones que se desean controlar y vincularlas a su equivalente en el modelo humano.

Los archivos contenidos en este repositorio que son de relevancia para la solución a diseñar son:

- ***HumanStateProvider***, correspondiente al dispositivo encargado de resolver el problema de cinemática inversa dado un conjunto de mediciones generadas con sensores de movimiento y generar un estado cinemático. Este dispositivo corresponde al corazón del proyecto.
- ***RobotPositionController***, dispositivo diseñado para controlar los movimientos y posición de

¹Repositorio disponible en <https://github.com/robotology/human-dynamics-estimation>

²URDF viene de las siglas en inglés *Unified Robotics Description Format*.

un robot dado un estado cinemático. Gracias a este dispositivo se puede controlar diferentes tipos de robots humanoides sin necesidad de modificar el *HumanStateProvider*.

- **Archivos de configuraciones de dispositivos.** Tal como indica su nombre, en estos archivos se definen los dispositivos a iniciar con sus respectivas configuraciones iniciales, de modo que el sistema total se ponga en marcha.

Otro punto característico del trabajo ya existente, es que la información de entrada al controlador proviene de sensores utilizando el formato y las clases definidas en *iWear YARP Interface*³, y el intercambio de información se realiza a través de una red a la que tanto los *SensorWrapper* como el controlador están conectados. Esta información es recibida por el controlador a través de puertos definidos en el archivo de configuraciones.

4.2. Diseño y estructura de la solución

4.2.1. Nuevo enfoque: sin sensores, sólo visión

El objetivo de este trabajo es lograr la teleoperación de un robot humanoide utilizando únicamente información generada con una cámara de video fija, eliminando completamente el uso de sensores cinemáticos y de profundidad. Para lograr esto, se debe incluir en el sistema un modelo basado en visión computacional que sea capaz de identificar un cuerpo humano en una imagen, video grabado o transmisión en directo, y que a partir de esta permita obtener información sobre la posición y dinámica de sus articulaciones y extremidades. Con esto, se reemplazaría la información que anteriormente se generaba utilizando sensores, eliminando así la dependencia del sistema de estos.

Otro punto a considerar es la velocidad de inferencia del modelo a utilizar. Dado que se espera que el control del robot sea en tiempo real, la velocidad de inferencia debe ser tal que permita que el robot genere movimientos fluidos y que le permita reproducir las poses en un tiempo mínimo. Además, en el caso de que se necesiten computar valores no proporcionados por el modelo de visión, estos deben realizarse de forma optimizada para no retrasar el sistema.

Respecto al resto del sistema (red y controladores), se espera mantener la estructura de estos, modificando únicamente las partes relacionadas a la lectura de datos generados por sensores y reemplazándolos por lectura de datos provenientes del modelo de visión. Para procurar mantener al mínimo los cambios realizados, se buscará replicar las clases definidas encargadas de procesar los datos provenientes de los sensores.

4.2.2. Elementos y estructura del nuevo sistema

Teniendo en consideración todos los antecedentes previos, se determina que la solución general debe constar de 4 secciones principales:

- **Detección e identificación de poses:** en esta etapa se realizará la identificación de la pose y movimientos presentes en el video o transmisión, y se obtendrá los datos que el modelo ya entrenado de visión sea capaz de generar. Los valores obtenidos se pasarán a un bloque de

³Más información en <https://github.com/robotology/wearables>

Transformación de datos

- **Transformación de datos:** este bloque se encargará de realizar los cálculos y aproximaciones necesarias para que los paquetes de datos contengan toda la información cinemática necesaria para que el controlador resuelva el problema de cinemática inversa. El algoritmo de visión a utilizar tiene una cantidad definida y limitada de *links* y *joints* que puede identificar, por lo que será necesario realizar aproximaciones geométricas y/o matemáticas de la información faltante.
- **Control:** este bloque contendrá los archivos y dispositivos necesarios para resolver de forma optimizada el problema de cinemática inversa y realizar el *retargeting* de la pose tanto al modelo humano como a modelos de robots humanoides. Además, se encargará de levantar el servidor, junto con la red y los puertos para la comunicación entre los distintos bloques del sistema.
- **Validación:** en esta etapa se evaluará cualitativa y cuantitativamente los resultados obtenidos tanto para los diferentes sub sistemas que compongan la solución como para el sistema completo funcionando. Se compararán los resultados obtenidos al final del sistema y se compararán con la entrada.

4.2.3. Alcance y limitaciones

El sistema, al basarse en visión computacional, está altamente condicionado por la visibilidad de la imagen capturada en tiempo real. Para garantizar el correcto funcionamiento, el operador debe permanecer completamente dentro del campo visual de la cámara, mostrando todas sus extremidades. En lo referente a la medición de la profundidad de la imagen para estimar la posición en el plano XY (paralelo al suelo), únicamente se empleará la estimación derivada del modelo de visión computacional. Es importante señalar que esta estimación posiblemente no alcance la misma precisión que la obtenida mediante sensores de profundidad, pero sí permitirá llevar a cabo los movimientos deseados.

En el contexto del alcance de la solución propuesta, se ha delimitado la teleoperación a la reproducción del movimiento de los brazos, piernas, pies y torso, omitiendo deliberadamente los movimientos de los dedos de cada mano por razones de simplificación del problema. Además, se aclara que el presente trabajo no abordará el problema asociado al desplazamiento del robot en el espacio (locomoción). Se establece que todos los experimentos a llevar a cabo en el robot real se realizarán con este sujeto a una base flotante, sin contacto con el suelo y sin requerir consideraciones sobre su estabilidad y equilibrio.

4.3. Solución propuesta y subsistemas

4.3.1. Diagrama de bloques

Dado que se ha establecido que la solución se compone de cuatro secciones fundamentales, se procede a definir el diagrama de bloques del sistema solución (ver Figura 4.1). Este diagrama está compuesto por cuatro subsistemas, cada uno definido y diseñado con el propósito de abordar las secciones mencionadas previamente. Estos subsistemas corresponden a *Visión*, *Transformación de datos*, *Control* y *Validación*.

A continuación, se procederá a detallar el alcance y los objetivos específicos de cada subsistema que integra la solución propuesta.

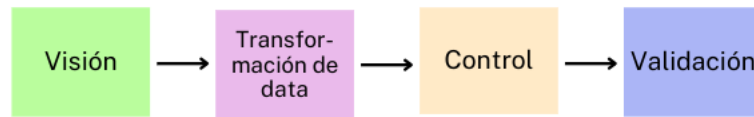


Figura 4.1: Diagrama de bloques del sistema propuesto.

4.3.2. Subsistema A: visión

El subsistema de visión constituye la etapa inicial encargada de capturar datos fundamentales para el desarrollo de la solución propuesta. Su función primordial reside en la obtención de información sobre la postura y ubicación de un individuo (en este caso, el operador) en el espacio 3D, a partir de un video en tiempo real. Este subsistema se encargará de generar las coordenadas tridimensionales (x, y, z) asociadas a cada articulación detectada en el cuerpo humano.

Para obtener las coordenadas de cada articulación, se utilizará *Mediapipe Pose*[11], algoritmo de detección de poses de código abierto que permite obtener las coordenadas de posición para 33 puntos o *joints* del cuerpo humano. De los 33 puntos generados por *Mediapipe*, sólo se utilizarán 15 de los puntos generados, correspondientes a los definidos para mapear el movimiento de las extremidades, torso y cabeza. Específicamente, los puntos seleccionados y su respectivo índice corresponden a:

- **Nariz (0)**, el cual permitirá mapear los movimientos de la cabeza.
- **Hombro izquierdo (11)** y **hombro derecho (12)**, para mapear el movimiento de los hombros.
- **Codo izquierdo (13)** y **codo derecho (14)**, para mapear los movimientos del brazo y su posición.
- **Muñeca izquierda (15)** y **muñeca derecha (16)**, para mapear el movimiento de las manos.
- **Cadera izquierda (23)** y **cadera derecha (24)**, para mapear el movimiento de la pelvis y estimar su longitud.
- **Rodilla izquierda (25)** y **rodilla derecha (26)**, para mapear el movimiento de las piernas.
- **Tobillo izquierdo (27)** y **tobillo derecho (28)**, para mapear el movimiento de los pies.
- **Índice del pie izquierdo (31)** e **índice del pie derecho (32)**, para mapear la posición y orientación del pie.

En la Figura 4.2 se muestran, en cuadros amarillos, los puntos seleccionados por este subsistema. Los puntos pertenecientes a expresiones faciales no fueron considerados ya que el reconocimiento y mapeo de expresiones faciales no es parte de este problema de investigación. En cuanto a las manos, se decide considerar únicamente la articulación de la muñeca.

Ya conocida la ubicación y el punto mapeado, se pasa al subsistema B para el cálculo del resto de valores cinemáticos.

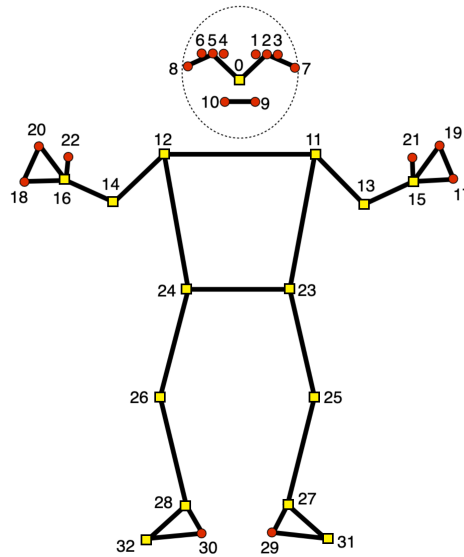


Figura 4.2: Puntos o *landmarks* seleccionadas para el sistema.

4.3.3. Subsistema B: transformación de data

Para poder lograr el control del robot se deben trabajar los valores generados por el subsistema de visión. Específicamente, las mediciones a generar por cada punto detectado son: posición, orientación, velocidad lineal y velocidad angular. Con estos valores, se generará un vector de datos que será enviado como mensaje al sistema de control. Para su facilitar su integración con el sub sistema de visión y el susbsistema de control, este se desarrollará utilizando *Python*, ya que el sub sistema de visión también se encuentra desarrollado en *Python*, y el sub sistema de control se comunicará a través de una red YARP, a la cual es posible acceder utilizando *bindings*⁴ creados específicamente para *Python*.

La posición (x, y, z) es entregada por el sistema de visión, y a partir de esta se estima el resto de valores.

Estimación de los ángulos *roll*, *pitch* y *yaw*

Para obtener los ángulos *roll*, *pitch* y *yaw* de cada articulación se utiliza el método propuesto por Temuge Batpurev [12]. Este método utiliza los (ya conocidos) valores de posición de cada articulación y considera como vectores las extremidades o *links* entre articulaciones. Con esto en mente, se puede trabajar el problema de estimación de ángulos como un problema de rotación de vectores: los ángulos *roll*, *pitch* y *yaw* se pueden obtener de la matriz de rotación que representa la rotación necesaria para que un *link padre* alcance la posición de el *link hijo* utilizando como eje de rotación la articulación o *joint* que los une.

El método mencionado anteriormente establece un procedimiento de tres etapas que para obtener una aproximación de los ángulos de cada *joint*, las cuales son: obtener la matriz de rotación que permita rotar un vector padre para alcanzar la posición del vector hijo, descomponer la matriz de

⁴Un *binding* es una aplicación que crea una interface para que que un lenguaje de programación pueda utilizar librerías de otro lenguaje. En este caso, YARP está asociado al lenguaje C++, pero gracias a los *bindings* es posible utilizar sus funciones en *Python*.

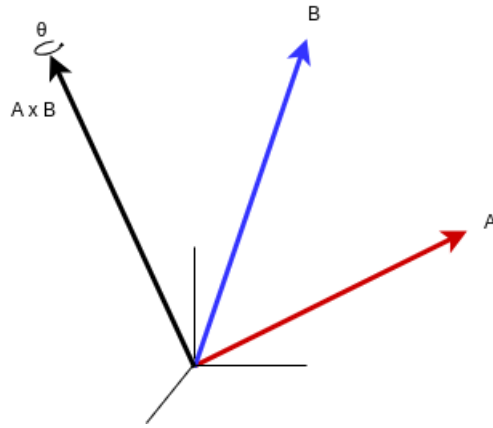


Figura 4.3: Dirección normal definida y rotación para obtener vector B a partir de A.

rotación a una secuencia de rotación en base a los ejes principales y definir una pose cero para usar como referencia (en este caso, la pose T).

Para obtener la matriz de rotación que permita rotar un vector a otro, se parte de la base de que se puede rotar un vector A hacia B definiendo una dirección normal dada por $A \times B$ y luego rotando a lo largo de este nuevo eje (ver Figura 4.3). Considerando esto, la estrategia propuesta busca cambiar las coordenadas al sistema definido por A , B , $A \times B$, rotar en esta base y luego regresar a la base original.

Entonces, se define una nueva base haciendo que A sea un vector unitario y tomando la proyección ortogonal de B sobre A como el segundo eje. El tercer eje se define simplemente como $A \times B$ normalizado. Con esto se escribe la matriz de rotación de cambio de base: cuando las coordenadas estén alineadas, simplemente se debe rotar a lo largo de la dirección $A \times B$, que está dada por la matriz de rotación R_z . Sin embargo, no es necesario conocer el ángulo de rotación, ya que $\cos(\theta)$ y $\sin(\theta)$ son simplemente el producto punto y el producto cruz de los vectores normalizados A y B , respectivamente, por lo que se puede obtener la matriz de rotación.

Con la matriz de rotación ya generada, esta se descompone teniendo en cuenta el orden de rotación a través de los ejes. Los ángulos de rotación se pueden calcular expandiendo la matriz como se muestra en las Ecuaciones 4.1 y 4.2.

$$\begin{aligned}
 R &= R_z(\theta_z)R_x(\theta_x)R_y(\theta_y) \\
 &= \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 \\ \sin \theta_z & \cos \theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{bmatrix} \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y \\ 0 & 1 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y \end{bmatrix} \quad (4.1)
 \end{aligned}$$

$$R = \begin{bmatrix} \cos \theta_z \cos \theta_y - \sin \theta_z \sin \theta_x \sin \theta_y & -\sin \theta_z \cos \theta_x & -\cos \theta_z \sin \theta_y + \sin \theta_z \sin \theta_x \sin \theta_y \\ \sin \theta_z \cos \theta_y + \cos \theta_z \sin \theta_x \sin \theta_y & \cos \theta_z \cos \theta_x & \sin \theta_z \sin \theta_y - \cos \theta_z \sin \theta_x \sin \theta_y \\ -\cos \theta_x \sin \theta_y & \sin \theta_x & \cos \theta_x \cos \theta_y \end{bmatrix} \quad (4.2)$$

En particular, para encontrar los ángulos de rotación de cada eje se usa:

$$\theta_x = \arctan\left(\frac{R_{[2,1]}}{\sqrt{R_{[2,0]}^2 + R_{2,2}^2}}\right) = \arctan\left(\frac{\sin(\theta_x)}{\sqrt{(-\cos(\theta_x)\sin(\theta_y))^2 + (\cos(\theta_x)\cos(\theta_y))^2}}\right) \quad (4.3)$$

$$\theta_y = \arctan\left(\frac{-R_{[2,0]}}{R_{[2,2]}}\right) = \arctan\left(\frac{\cos(\theta_x)\sin(\theta_y)}{\cos(\theta_x)\cos(\theta_y)}\right) \quad (4.4)$$

$$\theta_z = \arctan\left(\frac{-R_{[0,1]}}{R_{[1,1]}}\right) = \arctan\left(\frac{\sin(\theta_z)\cos(\theta_x)}{\cos(\theta_z)\cos(\theta_x)}\right) \quad (4.5)$$

Lo que nos permite encontrar los tres ángulos deseados equivalentes a *roll*, *pitch* y *yaw*.

Finalmente, se define la pose 0 de la cual se determinarán las rotaciones del esqueleto. La pose definida corresponde a la pose T, con los sistemas de referencia definidos como se muestra en la Figura 4.4. Las extremidades finales tales como muñecas o pies no necesitan definir su sistema de referencia ya que la posición de estas articulaciones usa la matriz de rotación del *link* padre. En este caso se define el centro de las caderas como la articulación base, a partir del cual se definirán el resto de articulaciones y extremidades. Se normalizan los largos de las extremidades a partir del largo de las caderas, y el sistema de referencia del cuerpo completo tendrá su centro en la articulación base.

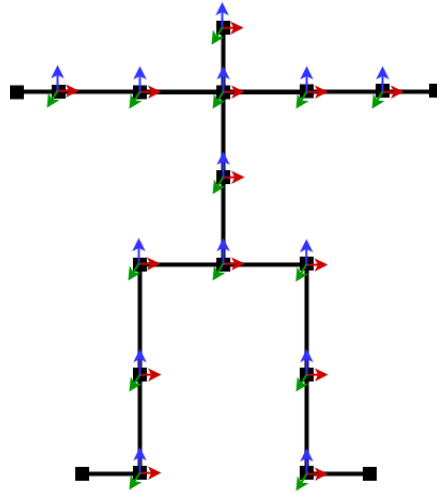


Figura 4.4: Modelamiento del cuerpo humano a base de *links* y *joints* en la pose T.

En cuanto a los valores de velocidad lineal y angular a generar, se crean dos opciones: utilizar una velocidad fija predefinida o estimar las velocidades a partir de la imagen. La primera opción es simple y efectiva, ya que reduce los cálculos a computar para generar el vector total de datos y permite limitar la velocidad de movimiento, asegurando la estabilidad y seguridad de la teleoperación. Sin embargo, la segunda opción ofrece mayor exactitud en la tarea de teleoperación en términos de replicar los movimientos deseados.

Para utilizar una velocidad pre definida basta con insertar en el vector de datos a enviar la velocidad deseada. Para obtener la velocidad a partir de la imagen, se puede estimar utilizando las

variación *frame a frame* de cada *keypoint* utilizando la Ecuación 4.6 , donde $\dot{\vec{p}}_i$ corresponde a la velocidad en el *frame* actual, \vec{p}_i corresponde a las coordenadas de posición (x, y, z) del *keypoint* en el *frame* actual, \vec{p}_{i-1} es la posición del *keypoint* en el *frame* anterior, y $1/\text{FPS}$ corresponde al periodo, el cual corresponde a la diferencia de tiempo entre los *frames*, también conocido como el ratio *FPS: frames per second* o “cuadros por segundo”. Esta ecuación es utilizada tanto para calcular la velocidad lineal de los *keypoints* de como también para calcular la velocidad angular de las articulaciones.

$$\dot{\vec{p}}_i = \frac{\vec{p}_i - \vec{p}_{i-1}}{1/\text{FPS}} \quad (4.6)$$

Ya con todas las aproximaciones realizadas, se genera el vector de datos a enviar al controlador. Para el envío de este se crea un puerto *YARP* para escritura de mensajes y este se conecta al puerto de lectura expuesto por el controlador. Como este subsistema se encuentra escrito en *Python*, se desarrolla una función en *Python* que permita enviar los mensajes y conectarse al servidor *YARP* ya levantado.

```
import yarp
def send_yarp_msg(str:msg, str:output_port):
    """ send message as string through yarp port """
    # Creates Bottle for sending message
    message = yarp.Bottle()
    # Adds message
    message.addString(str(msg) )
    # Send the message through the output port
    output_port.write(message)
```

Para poder utilizar esta función, se debe haber previamente iniciado y conectado el puerto de escritura al de lectura, tal como se muestra a continuación.

```
# Init YARP network in python
yarp.Network.init()
# Creates and open writer port
output_port = yarp.Port()
output_port.open("/wport")
# Connect writer port to reader port
yarp.Network.connect("/wport", "/rport")
```

Con esto, ya se cuenta con todos los datos necesarios para pasar al subsistema de control. La totalidad del código desarrollado para los subsistemas A y B se encuentra en el repositorio de *GitHub iCub-vision-teleoperation*, disponible en <https://github.com/f41lfaro/iCub-vision-teleoperation>.

4.3.4. Subsistema C: control

Una vez generados todos los valores cinemáticos y de posición necesarios para replicar la pose humana, estos se reciben a través de un puerto de lectura expuesto en una red *YARP*. El subsistema

de control tiene como propósito recibir y procesar los paquetes de datos provenientes de los subsistemas previos, con el fin de generar las posiciones y velocidades objetivos específicas para las diversas articulaciones del robot tal que, la configuración o posición resultante de este se asemeje lo más posible a la pose del operador, considerando las limitaciones físicas del robot.

La entrada al subsistema de control corresponde a un paquete (por cada *frame* de video) de datos, el cual contiene únicamente una línea de texto compuesta de 195 valores decimales separados por espacios. Cada paquete de datos p_i con información de pose para el instante i se compone de sub-paquetes p_{ij} , los que contienen la información cinemática y de posición de cada *joint* j a mapear. Para el caso del robot *iCub* se desea controlar en total 15 *joints* de este, por lo que un paquete de datos para el instante i de grabación tendrá la estructura mostrada en la Ecuación 4.7.

$$p_i = p_{i0} p_{i1} p_{i2} \dots p_{i15} \quad (4.7)$$

Más específico, cada sub-paquete de datos contendrá 13 valores con la siguiente estructura:

$$p_{ij} = [x_{ij} \ y_{ij} \ z_{ij} \ v_{ij}^x \ v_{ij}^y \ v_{ij}^z \ \dot{\theta}_{ij} \ \dot{\phi}_{ij} \ \dot{\psi}_{ij} \ q_{ij}^1 \ q_{ij}^2 \ q_{ij}^3 \ q_{ij}^4]$$

en donde

- p_{ij} corresponde al paquete de datos del instante i para la articulación j
- x_{ij} , y_{ij} y z_{ij} corresponden a las coordenadas espaciales (x, y, z) (posición) del instante i para la articulación j
- v_{ij}^x , v_{ij}^y y v_{ij}^z corresponden a la velocidad lineal aproximada en el instante i para la articulación j
- $\dot{\theta}_{ij}$, $\dot{\phi}_{ij}$ y $\dot{\psi}_{ij}$ corresponden a las velocidades angulares en los ángulos *roll*, *pitch* y *yaw* para el instante i en la articulación j .
- q_{ij}^1 , q_{ij}^2 , q_{ij}^3 y q_{ij}^4 corresponden a los valores del *quaternion* (orientación) en el instante i para la articulación j , donde q_{ij}^1 corresponde a la parte real y q_{ij}^2 , q_{ij}^3 , q_{ij}^4 a la parte imaginaria.

La salida del controlador corresponde a los ángulos *roll*, *pitch*, *yaw* y la velocidad con la que se moverán los actuadores presentes en las distintas articulaciones del robot para que su postura final coincida con la pose realizada por el operador. Puede resultar algo confuso que la parte de la entrada a este subsistema contenga los ángulos estimados de la pose que realiza el operador humano y la salida corresponda también a ángulos y velocidades que determinan la pose objetivo, pero la diferencia en la entrada y salida radica en que esta se adecua a la geometría y estructura del robot.

Como ha sido mencionado anteriormente, el controlador desarrollado está basado en la solución propuesta por el artículo [10] y el código disponible en el repositorio *human-dynamics-estimation*. A partir de este proyecto se modifican la estructura, dispositivos y entradas de forma que ahora lea los paquetes de datos previamente mencionados.

El controlador cuenta con 4 dispositivos que entran en funcionamiento: *Transform Server*, *Human State Provider*, *Robot State Visualizer* y *Robot Position Controller*. A continuación se detallará el funcionamiento de cada dispositivo y las principales modificaciones realizadas respecto del proyecto original.

Transform Server

Transform Server es el primer dispositivo en iniciarse. Este dispositivo en realidad no es desarrollado dentro del proyecto *Human Dynamisc Estimation*, sino que corresponde a uno de los requisitos de instalación para el funcionamiento del proyecto. El componente *Transform Server* se encarga de manejar tareas de transformaciones relacionadas al manejo de datos espaciales. Este dispositivo es de utilidad para manejar transformaciones entre distintos sistemas de referencia y permite convertir la data espacial de un marco de referencia a otro. Para este proyecto, el *Transform Server* ayudará a exponer los resultados obtenidos desde el *Human State Provider*, para que pueda ser leída por los visualizadores y simuladores para la validación.

Human State Provider

Human State Provider es el dispositivo principal del controlador. En este se generan los valores objetivos de ángulos *roll*, *pitch*, *yaw* y velocidades a las cuales se moverán los actuadores de las articulaciones del robot. Originalmente la entrada de este dispositivo provenía de los sensores de movimiento integrados en el traje *XSens Suit*, la cual era procesada y remapeada por el dispositivo *iWear Remapper*. Sin embargo, con la eliminación de los sensores de este sistema en la solución propuesta, estos dispositivos han dejado de ser indispensables. Ahora, los datos de entrada son adquiridos directamente por el *Human State Provider* a través de su puerto de lectura expuesto en la red YARP.

Al iniciarse el dispositivo y luego de cargar todos los parámetros y configuraciones definidas en el archivo de configuraciones (extensión *.xml*), este entra en su estado inicial 0, correspondiente a *Empty Data*. En este estado, el dispositivo queda a la espera de un vector de datos.

Una vez es recibido el vector de datos, el dispositivo entra al estado *Reading Data* y en este estado se realizará una verificación del vector para asegurar su conformidad con las características requeridas, en donde se chequea que este contenga efectivamente 195 valores correspondiente a la data para 15 *joints*. Si el vector no cumple con este requisito, el *HSP* volverá al estado *Empty Data*. Si el vector cuenta con los valores para los 15 actuadores, entonces avanza al siguiente estado de *New Data*.

En el estado *New Data* se actualizan los valores de la pose objetivo utilizando la función *UpdateVisionTargets()*, para la cual se debe resolver el problema cinemático. El vector recibido es dividido en secciones, una para cada actuador y para cada sección se almacena los valores de posición, velocidad lineal, velocidad angular y rotación. Una vez actualizados todos los *targets*, el controlador automáticamente resuelve los problema de cinemática inversa y velocidad inversa y genera los valores de salida que permitirán modificar la posición del robot. En la Figura 4.5 se ilustra el diagrama de flujo del dispositivo *HumanStateProvider*.

Ya teniendo las salida del *HSP*, se debe escoger si se desea levantar el visualizador *Human State Visualizer* o si se desea ya operar el robot real con *Robot Position Controller*. Ambos dispositivos no pueden funcionar simultáneamente, debido a que la infraestructura del proyecto no permite que más de un dispositivo esté conectado al puerto de salida del controlador (*Human State Provider*).

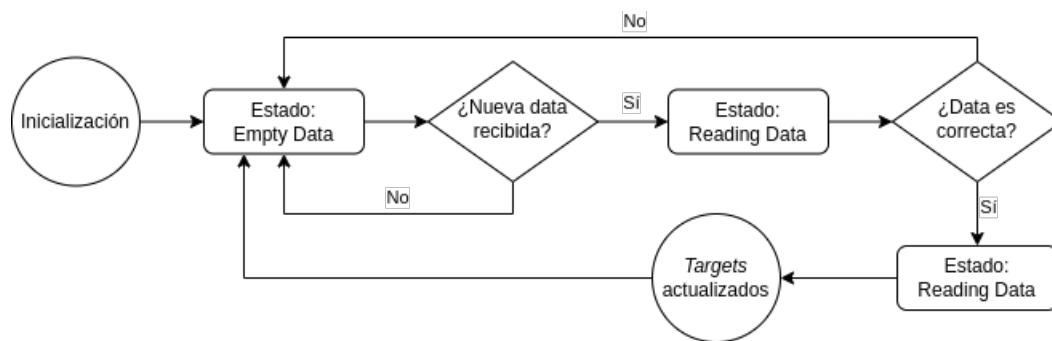


Figura 4.5: Diagrama de flujo del dispositivo *HumanStateProvider*.

Human State Visualizer

Human State Visualizer es un visualizador basado en la biblioteca *iDyntree* diseñado para la observación y análisis del comportamiento del robot durante el proceso de teleoperación. Este software permite la carga y visualización del modelo *urdf* asociado al robot en cuestión. El visualizador permite la visualización dinámica de la posición, orientación y velocidades de las extremidades del robot conforme se mueven para alcanzar la pose objetivo. Sin embargo, es importante destacar que este visualizador no incorpora consideraciones acerca de las interacciones del robot con su entorno, como el suelo y la gravedad, ni aborda las propiedades físicas del robot, incluyendo el equilibrio estático y dinámico, así como las limitaciones físicas inherentes al rango de movimiento máximo de los actuadores en las articulaciones.

El *Human State Visualizer* es una herramienta adecuada para la validación preliminar de los resultados obtenidos y la visualización de las poses alcanzadas durante la teleoperación. No obstante, una vez completada esta fase exploratoria inicial, resulta necesario analizar los resultados utilizando un simulador que se asemeje más al comportamiento del robot en condiciones reales, considerando de manera más precisa las interacciones ambientales y las limitaciones físicas del sistema. El simulador que sí considera todas estas interacciones y limitaciones es *Gazebo*, el cual se utiliza mediante el dispositivo *Robot Position Controller*.

Robot Position Controller

El *Robot Position Controller* constituye el dispositivo encargado de modificar la posición del robot físico. Este controlador se encuentra interconectado a la interfaz *Robot Control Board*, la cual posibilita la manipulación de los actuadores del robot, ya sea del robot físico como de la simulación del mismo en *Gazebo*. El *Robot Position Controller* recibe las salidas producidas por el dispositivo *Human State Provider (HSP)*, permitiendo así la iniciación del movimiento del robot con el fin de alcanzar la posición objetivo determinada. Además de esta función, este dispositivo facilita el registro de los movimientos ejecutados por los actuadores mediante *encoders* presentes en el robot. Este registro posibilitará una evaluación posterior, permitiendo comparar entre la pose final generada y la posición inicial deseada. Cabe señalar que, a diferencia del *Human State Provider*, el *Robot Position Controller* no genera valores que describan una posición para el robot; en su lugar, emplea los valores preexistentes para dirigir y controlar los movimientos del robot.

Al igual que para los subsistemas A y B, los dispositivos modificados y sus respectivos archivos de configuración para su uso se encuentran en el repositorio de *GitHub iCub-vision-teleoperation*,

disponible en <https://github.com/f41faro/iCub-vision-teleoperation>.

4.3.5. Subsistema D: validación

En este subsistema se hará uso de los *encoders*, que consisten en sensores presentes en los actuadores de las articulaciones del robot iCub y que permiten tener conocimiento de la posición real alcanzada de las articulaciones y extremidades. Registrando el historial de movimiento de los *encoders* para un experimento en particular y registrando los vectores de datos de la pose objetivo generados por el subsistema B (transformación de data), es posible comparar la exactitud del movimiento replicado por el robot e identificar los limitantes de este.

El subsistema de validación, aunque constituye la última etapa definida en este proyecto, no forma parte integral de la solución en términos de ejecución de cálculos o generación de salidas directamente utilizables para el control del robot. No obstante, su importancia radica en la evaluación cuantitativa de los resultados derivados del proceso de *retargeting* de la postura. Los *encoders*, al proporcionar información precisa sobre la posición real alcanzada por las articulaciones y extremidades del robot, permitirá registrar el historial de movimiento durante un experimento específico y, utilizando como parámetro de comparación los vectores de datos generados por el subsistema B para las poses objetivo detectadas, se posibilita la comparación entre el movimiento replicado por el robot y la pose deseada.

Esta comparación cuantitativa permite identificar la precisión del movimiento reproducido por el robot, además de revelar posibles limitaciones o restricciones en su desempeño.

4.3.6. Sistema general

Ya que se encuentran definidos cada uno de los subsistemas que componen el sistema general, se procede a su evaluación. En el repositorio de *GitHub iCub-vision-teleoperation*⁵ se encuentra disponible la totalidad del proyecto realizado. En este se encuentran los requerimientos, códigos e instrucciones de uso de cada uno de los subsistemas desarrollados, junto con documentación del código desarrollado.

⁵Enlace: <https://github.com/f41faro/iCub-vision-teleoperation>.

Capítulo 5

Evaluación y análisis de resultados

5.1. Método de evaluación por subsistema

Para validar y evaluar el funcionamiento del esquema de teleoperación propuesto para el robot iCub, se llevarán a cabo diversos experimentos tanto en simulación como en el robot real. En una primera etapa, los experimentos se dividirán por subsistema y, posteriormente, se evaluará el rendimiento del sistema completo de teleoperación.

5.1.1. A: Visión

MediaPipe Pose en su documentación incluyen las métricas de evaluación obtenidas utilizando tres conjuntos de datos de validación para detectar el desarrollo de tres actividades físicas: Yoga, Baile y *HIIT*, los que contienen imágenes de una persona posicionada a una distancia de 2 a 4 metros de distancia de la cámara y en los que se evaluaron 17 puntos claves siguiendo la topología COCO [13]. Los resultados declarados por *Google* para los modelos utilizados por *Mediapipe Pose* se muestran en la Tabla 5.1, donde la precisión es calculada utilizando el Porcentaje de Puntos Correctos (o PCK en inglés) donde un punto se considera correctamente detectado si el error o distancia euclidiana 2D entre la ubicación del punto detectado y el punto etiquetado en la imagen de referencia utilizada para evaluar es menor al 20 % del tamaño del torso de la persona detectada [13].

Tabla 5.1: Precisión de detección de pose de BlazerPose para distintos conjuntos de datos. Valores declarados por Google [1].

Method	Yoga mAP	Yoga PCK@0.2	Dance mAP	Dance PCK@0.2	HIIT mAP	HIIT PCK@0.2
BlazePose GHUM Heavy	68.1	96.4	73.0	97.2	74.0	97.5
BlazePose GHUM Full	62.6	95.5	67.4	96.3	68.0	95.7
BlazePose GHUM Lite	45.0	90.2	53.6	92.5	53.8	93.5
AlphaPose ResNet50	63.4	96.0	57.8	95.5	63.4	96.0
Apple Vision	32.8	82.7	36.4	91.4	44.5	88.6

Pose landmark versus Pose world landmark

Mediapipe Pose genera dos tipos de coordenadas normalizadas para los puntos detectados: *pose landmarks* y *pose world landmarks*. Ambos tipos inferencia generan los cuatro mismos valores: x , y , z y visibilidad, pero son calculados de forma diferente.

Los valores que contiene los *landmarks* corresponden a:

- x e y : coordenadas normalizadas (con valores entre 0.0 y 1.0) con respecto al ancho (x) y alto (y) de la imagen capturada por la cámara [14].
- z : medida de profundidad, utilizando el centro de la cadera de la persona como referencia. Mientras más cerca de la cámara se encuentre la persona, menor será el valor de profundidad [14].
- **Visibilidad**: valor que corresponde a la probabilidad de que un punto a detectar sea visible [14].

Mientras que los valores generados por los *world landmarks* corresponden a:

- x , y , z : coordenadas 3D del mundo real en metros, utilizando como origen el centro de las caderas [14].
- **Visibilidad**: valor que corresponde a la probabilidad de que un punto a detectar sea visible [14].

Para el desarrollo de este sistema, se escogerá trabajar con solo uno de estos dos tipos de valores entregados por *Mediapipe Pose*. Para saber cual tipo de punto utilizar, se desarrolla un primer método de evaluación con el objetivo de comparar el desempeño de ambos tipos de puntos.

El método de evaluación consiste en medir la variación entre las distancias de puntos vecinos. Los puntos al estar estimados sobre la silueta humana, no deberían presentar grandes variaciones de distancia con sus puntos *padres* o *hijos*, ya que una extremidad no variará su distancia de una articulación a otra en cosa segundos. Por ejemplo, el tamaño del antebrazo de una persona no cambiará de un segundo a otro, por lo que la distancia entre el punto de la muñeca y del codo detectados deberían mantener constante su distancia.

Esta métrica desarrollada permitirá estudiar las fluctuaciones en la distancia de los puntos detectados y, por lo tanto, las posibles deformaciones generadas en la silueta y pose detectada. Para calcular la diferencia entre los puntos se escoge una pose en la que todos los puntos a detectar sean claramente visibles y se obtienen las distancias entre los puntos vecinos. Este registro servirá como medición base para medir la variación durante el resto de las detecciones.

La distancia entre puntos es calculada utilizando la norma euclidiana, y el valor obtenido posteriormente es normalizado por el tamaño medido de la cadera (es decir, distancia de punto de cadera izquierda a cadera derecha). Para visualizar los resultados obtenidos, se considerarán las fluctuaciones en la distancia de los pares:

- hombro izquierdo - hombro derecho
- hombro izquierdo - codo izquierdo
- codo izquierdo - muñeca izquierda

- codo derecho - muñeca derecha
- cadera izquierda - rodilla izquierda
- hombro izquierdo - cadera izquierda
- hombro derecho - cadera derecha
- cadera derecha - rodilla derecha
- rodilla izquierda - tobillo izquierdo
- rodilla derecha - tobillo derecho

Se graficará la diferencia para los puntos detectados utilizando *pose landmark* y *pose world landmark*, y el tipo de punto que presente menores variaciones en las distancias de los pares de puntos definidos corresponderá al de mayor exactitud. Los resultados del experimento se encuentran en la Sección 5.3.1.

Tiempo de inferencia de la pose

Para que la solución propuesta cumpla con el requisito de ser en tiempo real, se evaluará el tiempo de inferencia del modelo para reconocer una pose. MediaPipe registra el tiempo de inferencia para distintos computadores y sus distintos modelos, los que se muestran en la Tabla 5.2. Debido a que el proyecto se trabajará utilizando un computador diferente a los registrados en la Tabla 5.2, se medirá el tiempo de inferencia en este. Los resultados de este experimento se registran en la Sección 5.3.1.

Tabla 5.2: Tiempos de inferencia para los modelos de detección de pose usados en MediaPipe Pose.

Modelo	Latencia en Pixel 3 TFLite GPU [ms]	Latencia en MacBook Pro (15-inch 2017) [ms]
BlazePose GHUM Heavy	53	38
BlazePose GHUM Full	25	27
BlazePose GHUM Lite	20	25

5.1.2. B: Transformación de data

Los experimentos a realizar para este subsistema buscarán evaluar el funcionamiento de sus dos tareas principales: la estimación de datos y el envío de estos usando puertos YARP. Para ambas pruebas se utilizará el *software* generado para el subsistema B en *Python*.

Estimación de ángulos

Para visualizar y evaluar los ángulos estimados, el procedimiento se basará en la representación gráfica de una pose humana en un sistema de coordenadas tridimensional cartesiano. Utilizando las coordenadas tridimensionales obtenidas con *Mediapipe Pose* y los ángulos estimados asociados a cada articulación del cuerpo, se generarán representaciones gráficas de la pose en el espacio tridimensional. Con esto, se podrá llevar a cabo una comparación visual entre los diferentes gráficos obtenidos, permitiendo así la evaluación y el análisis comparativo de las posturas representadas, y poder identificar similitudes, discrepancias o mejoras de la estimación.

El montaje experimental de este test contemplará el uso de un computador, una cámara web y un operador que realice movimientos frente a la cámara. En la imagen 5.1 se muestra la configuración experimental.

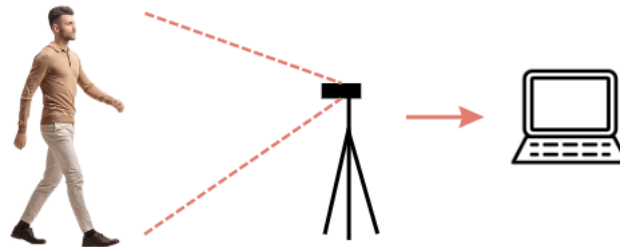


Figura 5.1: Montaje experimental para evaluación de estimación de ángulos.

Comunicación mediante red YARP

Uno de los desafíos de este proyecto es lograr la integración de todos los subsistemas de la solución, ya que el subsistema de *visión* (A) y *transformación de datos* (B) están escritos en diferente lenguaje de programación. Para evaluar la integración y comunicación entre estos, se testeará la generación y el envío de mensajes desde el subsistema B (*Python*) a un puerto de lectura levantado en un servidor YARP (*C++*). El procedimiento para verificar el envío de datos es:

1. En una primera terminal de comandos, se inicializa el servidor YARP escribiendo

```
yarpserver
```
2. En una segunda terminal de comandos, se crea un puerto de lectura de datos con el nombre definido en el código escrito para el sub sistema B (en este caso el nombre del puerto será */dataPort*). Para crear este puerto se escribe en la terminal:

```
yarp read /dataPort
```
3. En una tercera terminal de comandos se ejecuta el código correspondiente al subsistema B, escribiendo en la terminal ¹:

```
python3 webcam_vector_write_port_inv.py
```

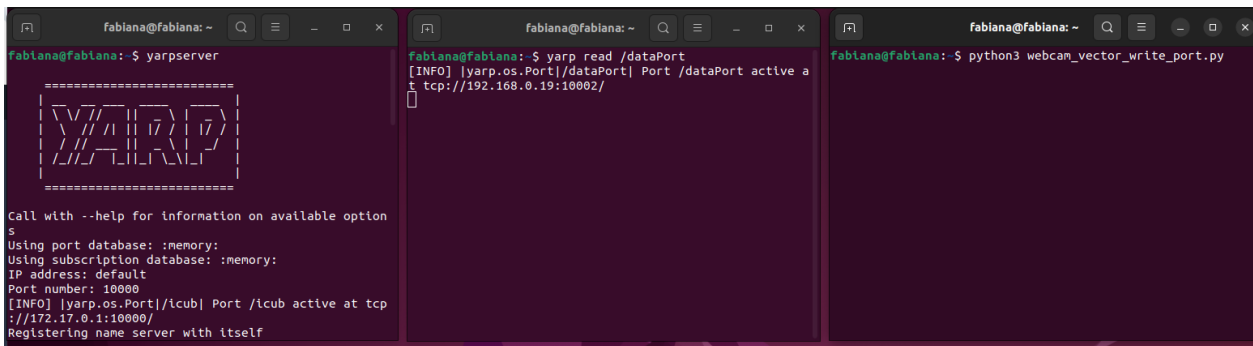
En la Figura 5.2 se ilustra cómo se deben ejecutar los comandos mencionados anteriormente en diferentes terminales.

Con esto, si los paquetes de datos generados por el subsistema B con correctamente enviados, deberían ser legibles y verse en el puerto de lectura */dataPort*.

Tiempo de generación de datos

Finalmente, se evaluará el tiempo que toma generar el vector de datos completo (considerando los valores de posición, orientación, velocidad lineal y angular) y enviarlo al servidor. Para esto, se registrarán los tiempos de 20 inferencias y se obtendrá un promedio de tiempo de inferencia. El resultado se considerará aceptable si el tiempo medido se encuentra por debajo de un segundo de desfase.

¹El código a ejecutar puede variar dependiendo de la versión del subsistema



```
fabiana@fabiana: ~  
fabiana@fabiana: $ yarpserver  
=====  
YARP  
=====  
Call with --help for information on available options  
Using port database: :memory:  
Using subscription database: :memory:  
IP address: default  
Port number: 10000  
[INFO] [yarp.os.Port]/lcub| Port /lcub active at tcp  
://172.17.0.1:10000/  
Registering name server with itself  
  
fabiana@fabiana: ~  
fabiana@fabiana: $ yarp read /dataPort  
[INFO] [yarp.os.Port]/dataPort| Port /dataPort active at  
tcp://192.168.0.19:10002/  
  
fabiana@fabiana: ~  
fabiana@fabiana: $ python3 webcan_vector_write_port.py
```

Figura 5.2: Montaje experimental para experimento de comunicación mediante red YARP.

5.1.3. C: Control

Los experimentos relativos a este subsistema se centran en la evaluación del desempeño del módulo de control tras la modificación de su código y la eliminación del empleo de sensores. Dichas pruebas se llevarán a cabo de manera previa a la conexión con el sistema completo y utilizando un simulador del robot, dado que este módulo determinará los movimientos del robot físico, y cualquier disfunción podría potencialmente comprometer la integridad estructural del robot.

Ejecución de pose objetivo

La evaluación consistirá en la transmisión de vectores de datos a través del puerto lector de datos `/dataPort`, generado al inicializar el controlador. Estos vectores contendrán información que represente poses y movimientos seguros para el robot (es decir, sin sobrepasar los límites físicos de movimiento de este). El envío de datos se realizará manualmente mediante la creación de un puerto de escritura de mensajes YARP, el cual se conectará al puerto de lectura de datos del controlador. Este procedimiento de creación y conexión manual de los puertos de datos será automatizado una vez que todos los subsistemas estén conectados e integrados.

El desarrollo de este experimento permitirá obtener 2 tipos de resultado: la visualización del movimiento a través de un gráfico 3D del robot y la simulación empleando *Gazebo*, la cual considera las propiedades físicas del robot y su interacción con el entorno, tales como el suelo, la gravedad y el equilibrio. En ambos casos, se llevará a cabo una evaluación mediante un análisis visual de los movimientos generados. Respecto a la simulación, que permite determinar numéricamente la posición final a través del uso de *encoders*, se realizará una comparación entre el vector de datos enviado y la pose final alcanzada.

Dentro de este experimento, se procederá al registro de los vectores de datos transmitidos. Los resultados comprenderán las imágenes tanto del visualizador como de la simulación generada. Además, se registrarán los valores adquiridos a partir de los *encoders* correspondientes a todas las articulaciones del robot. Se consideran exitosos los experimentos si los movimientos logrados corresponden a las articulaciones correctas, la pose generada es visualmente similar a la pose original del operados y si los valores angulares de las articulaciones del robot no difieren en más de un 20 % de la pose original.



Figura 5.3: Montaje experimental para experimentos del sistema completo con resultados de simulación.



Figura 5.4: Montaje experimental para experimentos del sistema completo con resultados en el robot real.

5.2. Método de evaluación del sistema general

Una vez concluida la evaluación individual de todos los subsistemas, se procede a integrar el sistema completo en un marco general para la teleoperación del robot. En los experimentos siguientes, se procederá a evaluar el rendimiento del sistema completo, tanto en entornos de simulación (*Gazebo*) como en el robot físico real. Los resultados obtenidos se registrarán mediante imágenes y grabaciones de video con el propósito de capturar la respuesta del sistema en tiempo real, permitiendo así una apreciación detallada de su funcionamiento.

5.2.1. Montaje experimental

El montaje experimental a utilizar se muestra en las Figuras 5.3 y 5.4 correspondientes a las pruebas a realizar en entornos de simulación y en el robot físico, respectivamente. En el entorno informático, se configurarán y ejecutarán todos los subsistemas previamente evaluados para conformar el sistema general.

En el caso de teleoperación local, tanto el sistema completo como el módulo del *Robot Control Board* serán iniciados y operados en un mismo computador. En el caso de la teleoperación remota, el sistema integral será iniciado y operado en el primer computador, mientras que el *Robot Control Board*, conectado al robot físico, se encontrará iniciado en el segundo computador.

5.2.2. Parámetros de evaluación

Para llevar a cabo una evaluación cuantitativa de los resultados generados por el sistema integrado, se empleará como medida métrica el cálculo del error existente entre los ángulos de las articulaciones que definen la pose inicial y la pose final lograda por el robot, tanto en el entorno

simulado como en el robot físico.

A través de la utilización de los *encoders* incorporados en la estructura del robot y del módulo de validación (subsistema D) de la solución, se registrarán los valores correspondientes a la posición final alcanzada. Se considerará que la pose obtenida es correcta si la diferencia entre los ángulos de la pose original y la pose final alcanzada se encuentra dentro del umbral de error es menor al 20 %. Esta medición establece un criterio de precisión, donde se evalúa la similitud entre la posición inicial y la posición final lograda por el robot, determinando así la exactitud del sistema implementado.

5.2.3. Teleoperación utilizando simulación

Como ya ha sido mencionado, los experimentos de simulación del comportamiento del robot se realizarán utilizando *Gazebo*. Antes de iniciar el sistema completo de teleoperación, se recomienda exponer los puertos de entrada del iCub simulado siguiendo los siguientes pasos:

1. En una primera terminal de comandos, se inicializa el servidor YARP escribiendo
`yarpserver`
2. En una segunda terminal de comandos, se inicia el entorno de simulación a utilizar, escribiendo en la terminal
`gazebo`
3. Una vez se abre la interfaz gráfica de *Gazebo*, se debe insertar en el “mundo” (espacio 3D mostrado por el simulador) uno de los modelos de robot disponibles en el menú. En este caso se trabajará con el iCub_v2.5 (ver Figura 5.5)

4. En una tercera terminal de comandos se inicializa la interfaz de control de los actuadores del robot (conocida como *Robot Control Board*) utilizando el comando

```
yarpmotorgui --robot icubSim
```

Se acepta la ventana mostrada en la Figura 5.6 y si el procedimiento se realiza de forma correcta, se abrirá la interfaz gráfica del *Control Board*, como se muestra en la Figura 5.7

5. Posteriormente, para iniciar el *TransformServer*, se debe abrir una nueva terminal desde el *path* donde se haya clonado el repositorio “human-dynamic-estimation-with-vision”, en particular desde la carpeta *conf/xml*. Desde esta ubicación, se ejecuta el comando

```
yarprobotinterface --config TransformServer.xml
```

lo que iniciará el dispositivo. Cuando la terminal retorne el mensaje `yarprobotinterface running happily` se puede pasar al siguiente paso.

6. Para iniciar el *HumanStateProvider* con las configuraciones para controlar un robot iCub, se debe abrir otra terminal desde el *path* donde se haya clonado el repositorio human-dynamic-estimation-with-vision, en particular desde la carpeta *conf/xml*. Desde esta ubicación, se ejecuta el comando

```
yarprobotinterface --config RobotStateProvider\_iCub2\_5\_sim.xml
```

lo que iniciará el dispositivo configurado con los parámetros del robot de simulación. El nombre del archivo indica que es para el iCub v2.5, pero las mismas configuraciones son compatibles con el robot iCub v2.7, por lo que no habrá problema al utilizarlo. Cuando la terminal retorne el mensaje `HumanStatePublisher: Run properly initialized` se procede con el siguiente paso.

7. En otra terminal abierta desde la ubicación *conf/xml*, se ejecuta el comando


```
yarprobotinterface --config RobotPositionController\_iCub\_sim.xml
```

 Como resultado de esto, el dispositivo tomará control de la venta de *yarpmotorgui*, lo que le permitirá modificar la posición de los ángulos del robot en gazebo.
8. Finalmente, en una nueva terminal abierta desde la carpeta donde se haya clonado el repositorio “vision-system-for-robot-teleoperation”, se ejecuta el archivo *webcam_vector_write_port_inv* utilizando el comando

```
python3 webcam\_vector\_write\_port\_inv.py
```

Esto activará la cámara y dará inicio a la detección de pose del operador e inferencia de ángulos. Al ejecutar este paso, el robot de la simulación comenzará a imitar las posiciones y movimientos del operador.

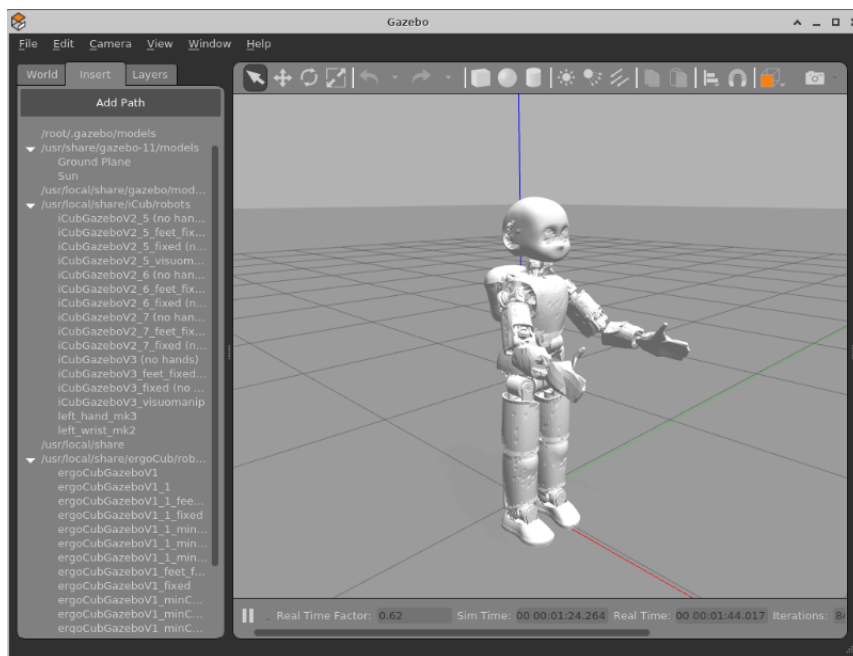


Figura 5.5: Configuración de *Gazebo* para los experimentos en simulación.



Figura 5.6: Menú de articulaciones del robot a operar.

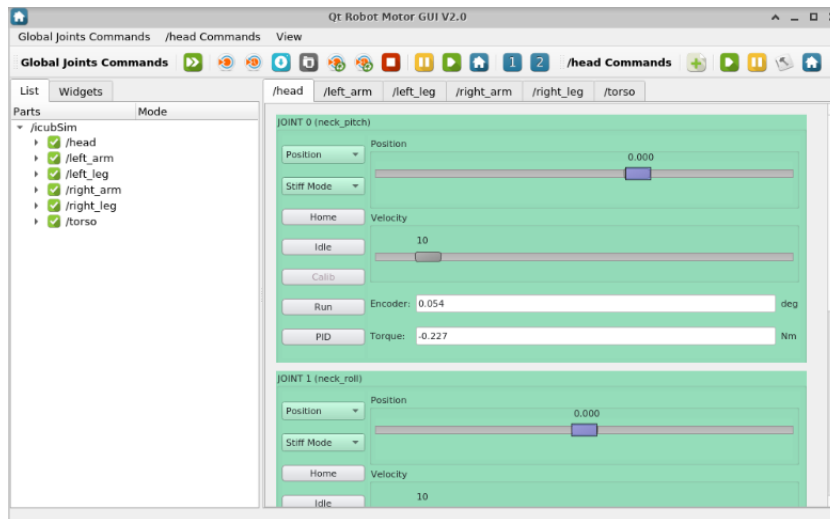


Figura 5.7: Interfaz gráfica del dispositivo *Control Board*.

5.2.4. Teleoperación utilizando el robot real - caso local

El código desarrollado para el control del iCub es compatible con el iCub v2.5, v2.6 y v2.7. En este caso, se utilizará el iCub v2.7. Para realizar las pruebas en el robot real de forma local (es decir, utilizando un solo computador conectado tanto a la *webcam* como al robot), se requiere de:

- Una *webcam* que permita capturar el cuerpo completo de un operador (si la cámara es externa, se recomienda utilizar un trípode para mantenerla fija).
- Un computador configurado para actuar como un servidor iCub y para ejecutar los sub sistemas de visión, transformación de datos y control.
- Un robot iCub junto con todos los aparatos necesarios para su funcionamiento: cable *Ethernet* para la comunicación entre el robot y el computador configurado como servidor iCub, cable de alimentación de energía, fuente de energía y soporte estructural del iCub.

A continuación se describen los pasos para ejecutar el sistema completo.

1. **Inicio de *yarpserver*:** Se abre una primera terminal, en la que se ejecuta el comando

```
yarpserver
```

como se muestra en la Figura 5.8.

```

icub@icubsrv: ~/Desktop
icub@icubsrv: ~/Desktop
* set "/tmp/port/1" process 31972
-> announce "/tmp/port/1"
* query "/tmp/port/1"
* query "/icubsrv"
* query "/icubsrv"
* unregister "/tmp/port/1"
* get "/icubsrv" yarprun
* register "...
* set "/tmp/port/1" offers http name_ser local tcp fast_tcp mcast
udp text text_ack bayer human mjpeg portmonitor priority rossrv shmem
tcpmux unix_stream xmlrpc
* set "/tmp/port/1" accepts http name_ser local tcp fast_tcp mcast
udp text text_ack bayer human mjpeg portmonitor priority rossrv shmem
tcpmux unix_stream xmlrpc
* set "/tmp/port/1" ips "127.0.0.1" "10.0.0.1" "10.31.82.69" ":::1"
"fe80::f372:b20:5783:d065%2" "fe80::e79f:5b70:e38a:6cf3%3"
* set "/tmp/port/1" process 31972
-> announce "/tmp/port/1"
* query "/tmp/port/1"
* query "/icub-head"
* unregister "/tmp/port/1"
icub@icubsrv:~/Desktop$ yarpsvr

```

```

icub@icubsrv: ~/Desktop
icub@icubsrv: ~/Desktop
+ set "/root" offers http name_ser local tcp fast_tcp mcast u
p text text_ack bayer human mjpeg portmonitor priority rossrv shm
em tcpmux unix_stream xmlrpc
+ set "/root" accepts http name_ser local tcp fast_tcp mcast u
dp text text_ack bayer human mjpeg portmonitor priority rossrv sh
mem tcpmux unix_stream xmlrpc
+ set "/root" ips "127.0.0.1" "10.0.0.1" "10.31.82.69" ":::1" "
fe80::f372:b20:5783:d065%2" "fe80::e79f:5b70:e38a:6cf3%3"
+ set "/root" process 36053
* register fallback mcast "224.2.1.1" 10000
+ set fallback offers http name_ser local tcp fast_tcp mcast u
dp text text_ack bayer human mjpeg portmonitor priority rossrv sh
mem tcpmux unix_stream xmlrpc
+ set fallback accepts http name_ser local tcp fast_tcp mcast
udp text text_ack bayer human mjpeg portmonitor priority rossrv s
hmem tcpmux unix_stream xmlrpc
+ set fallback ips "127.0.0.1" "10.0.0.1" "10.31.82.69" ":::1"
"fe80::f372:b20:5783:d065%2" "fe80::e79f:5b70:e38a:6cf3%3"
+ set fallback process 36053
* set "/root" nameserver "true"
Name server can be browsed at http://10.0.0.1:10000/
Ok. Ready!

```

Figura 5.8: Inicio de *yarpsvr*. A la izquierda, terminal antes de ejecutar el comando. A la derecha, terminal después de ejecutado el comando (se levanta el servidor *yarp*).

2. **Inicio de *yarpmanager*:** En una segunda terminal, se ejecuta el comando

`yarpmanager`

Esta acción abrirá una ventana con una interfaz gráfica que facilitará la conexión del robot *iCub* al servidor *yarp* previamente levantado (Figura 5.9).

```

icub@icubsrv: ~/Desktop
icub@icubsrv: ~/Desktop
/por/1 to /icub-head using tcp
[INFO] [yarp.os.tmpl.PortCoreOutputUnit] Removing output from /tmp/port/1 to /icub-head
[INFO] [yarp.os.tmpl.PortCoreInputUnit] Removing input from /tmp/port/1 to /icub-head
RESPONSE:
=====
exit OK
[INFO] [yarp.os.Port[/tmp/port/1]] Port /tmp/port/1 active at tcp://10.0.0.1:10002/
[INFO] [yarp.os.tmpl.PortCoreOutputUnit[/tmp/port/1]] Sending output from /tmp/port/1 to /icubsrv using tcp
[INFO] [yarp.os.tmpl.PortCoreInputUnit[/icubsrv]] Receiving input from /tmp/port/1 to /icubsrv using tcp
[INFO] [yarp.os.tmpl.PortCoreOutputUnit[/tmp/port/1]] Removing output from /tmp/port/1 to /icubsrv
[INFO] [yarp.os.tmpl.PortCoreInputUnit[/icubsrv]] Removing input from /tmp/port/1 to /icubsrv
[INFO] [yarp.os.Port[/tmp/port/1]] Port /tmp/port/1 active at tcp://10.0.0.1:10002/
[WARNING] Cannot connect to /icub-head
[ERROR] ClusterHead: port /icub-head is not responding
icub@icubsrv:~/Desktop$ yarpmanager

```

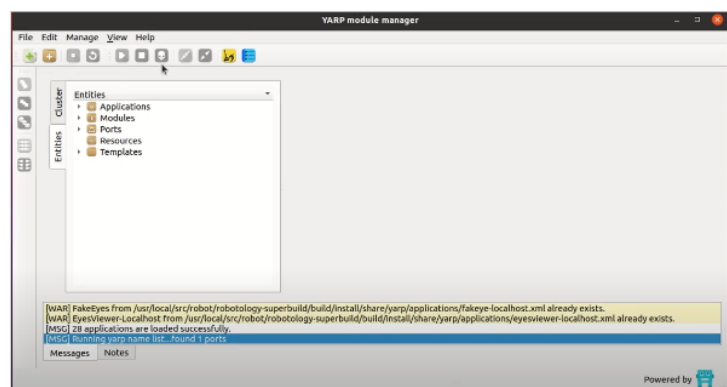


Figura 5.9: Inicio de *yarpmanager*. A la izquierda, terminal antes de ejecutar el comando. A la derecha, interfaz gráfica que se abre una vez ejecutado el comando anterior.

3. **Conexión a *iCub-head*:** Para poder utilizar el robot desde el computador, se debe conectar los nodos *icub-head* e *icubsrv*, tal como se muestra en la Figura 5.10. Cuando ambos nodos se encuentren conectados, el icono de *status* cambiará a color verde.

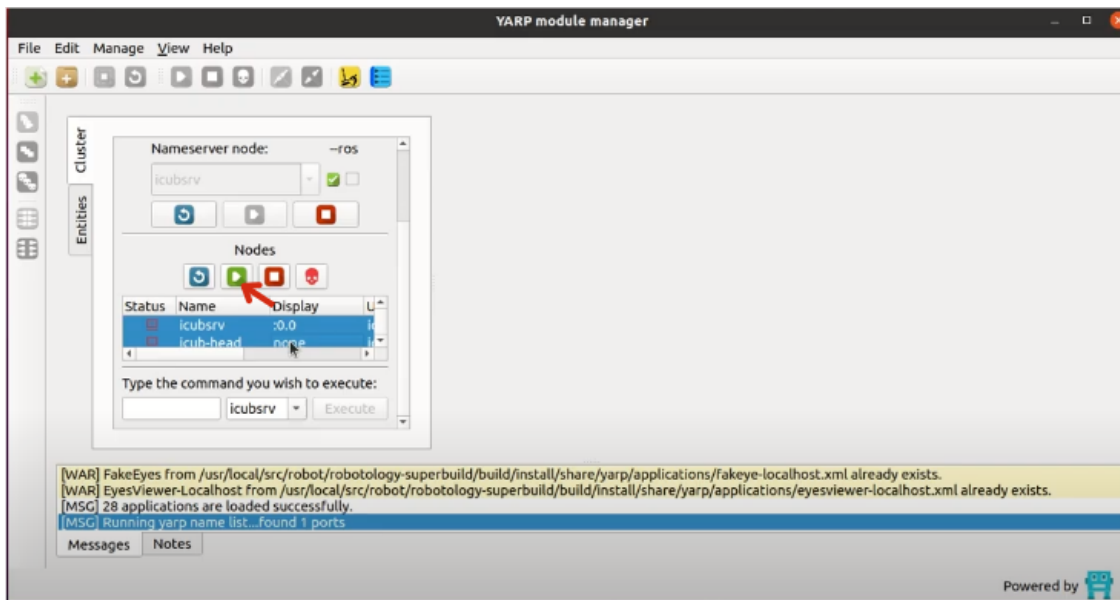


Figura 5.10: Conexión de nodos. Se seleccionan los nodos *icub-head* e *icubsrv* y se conectan utilizando el botón indicado por la flecha.

4. **Iniciar *yarprobotinterface*:** Se debe iniciar este módulo para tener acceso a los motores que controlan los movimientos del robot. Utilizando la interfaz levantada por *yarpmanager*, en la sección *Entities* ↔ *Applications* ↔ *1-iCubStartup* se selecciona el módulo *yarprobotinterface* y se inicializa presionando el botón indicado por el cursor en la Figura 5.11. Cuando se comience a ejecutar, el robot empezará a realizar movimientos correspondientes al chequeo inicial de sus articulaciones. Cuando termine este procedimiento, se debe continuar con el siguiente paso.

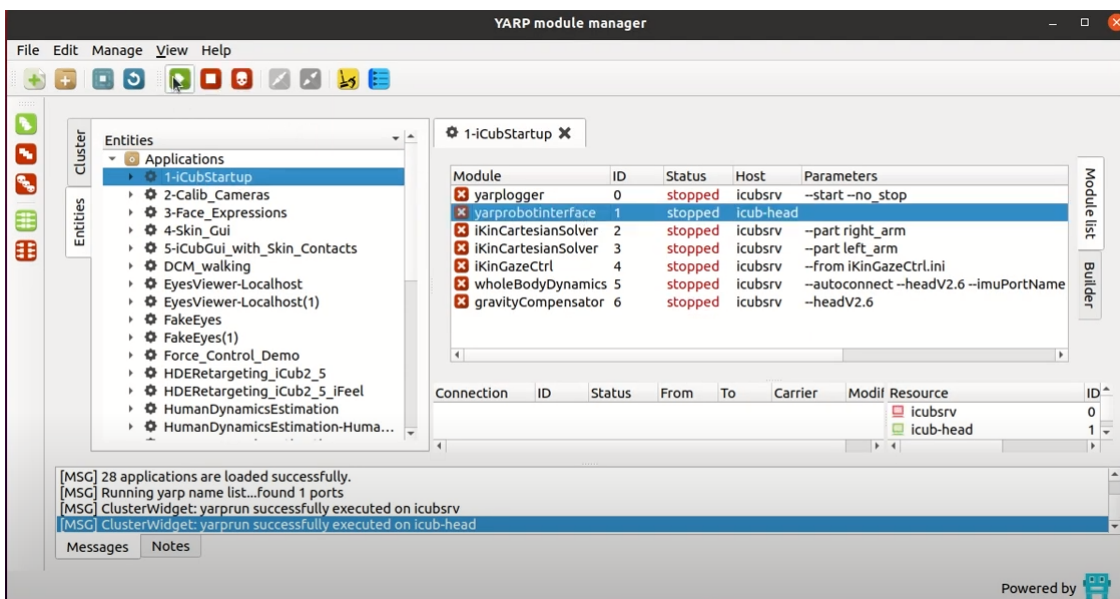


Figura 5.11: Conexión de nodos. Se seleccionan los nodos *icub-head* e *icubsrv* y se conectan utilizando el botón indicado por la flecha.

5. **Iniciar *yarpmotorgui*:** En una nueva terminal, se ejecuta el comando

```
yarpmotorgui --robot icub
```

Esta acción desplegará un menú de para escoger los subsistemas (extremidades y motores) con los que se trabajará. En este caso se seleccionan todas las opciones a excepción de */icub/face* (ver Figura 5.12), ya que para este trabajo no se utiliza ningún componente de la cara del robot. Una vez confirmado los subsistemas a utilizar, se abrirá una ventana como la mostrada en la Figura 5.13, en donde se podrá controlar la velocidad y los ángulos de posición de cada articulación del robot.

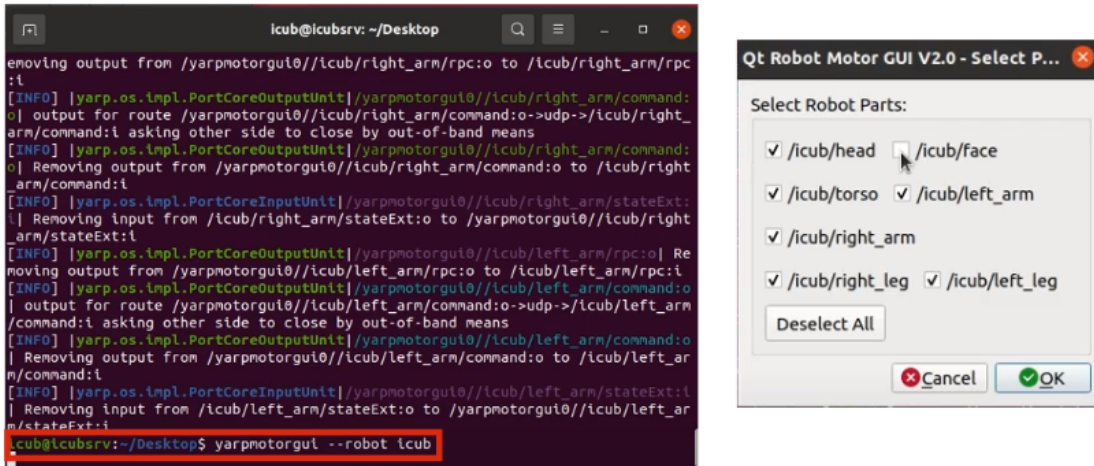


Figura 5.12: Inicio de *yarpmotorgui* para el robot iCub real.

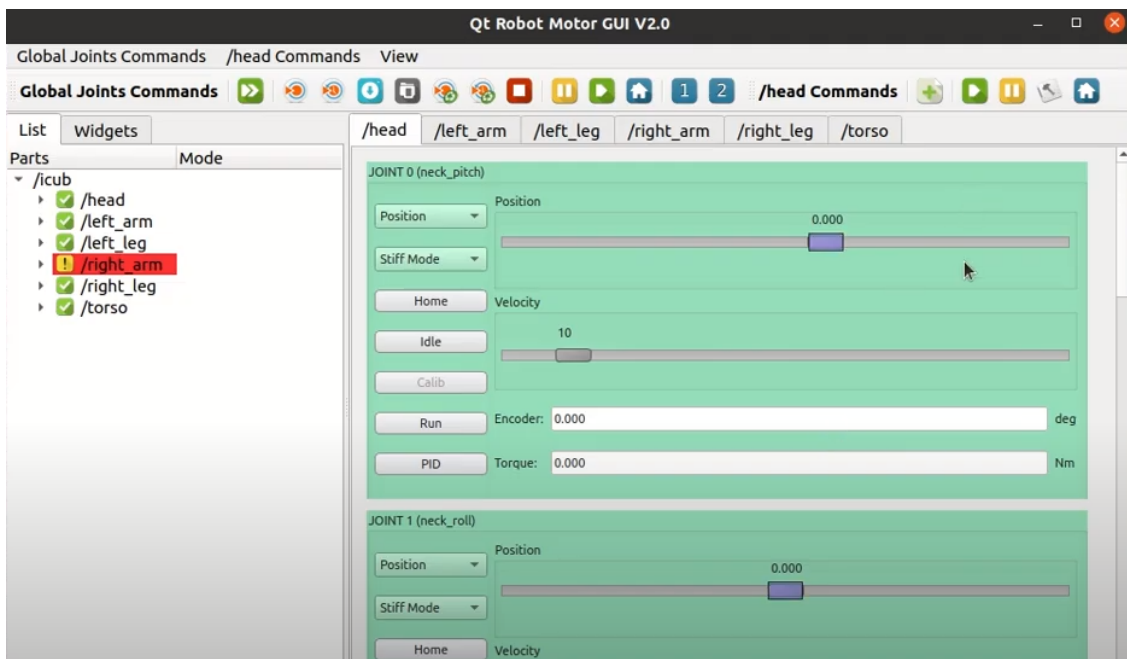


Figura 5.13: Interfaz gráfica de control de los motores y actuadores presentes en las articulaciones del robot.

6. **Iniciar *TransformServer*:** Para iniciar el *TransformServer*, se debe abrir una nueva terminal desde el *path* donde se haya clonado el repositorio “human-dynamic-estimation-with-vision”, en particular desde la carpeta *conf/xml/*. Desde esta ubicación, se ejecuta el comando

```
yarprobotinterface --config TransformServer.xml
```

lo que iniciará el dispositivo. Cuando la terminal retorne el mensaje `yarprobotinterface running happily` se puede pasar al siguiente paso.

7. **Iniciar *HumanStateProvider*:** Para iniciar el *HumanStateProvider* con las configuraciones para controlar un robot iCub, se debe abrir otra terminal desde el *path* donde se haya clonado el repositorio `human-dynamic-estimation-with-vision`, en particular desde la carpeta `conf/xml`. Desde esta ubicación, se ejecuta el comando

```
yarprobotinterface --config RobotStateProvider_iCub2_5.xml
```

lo que iniciará el dispositivo configurado con los parámetros del robot. El nombre del archivo indica que es para el iCub v2.5, pero las mismas configuraciones son compatibles con el robot iCub v2.7, por lo que no habrá problema al utilizarlo. Cuando la terminal retorne el mensaje `HumanStatePublisher: Run properly initialized` se procede con el siguiente paso.

8. **Iniciar *RobotPositionController*:** En otra terminal abierta desde la ubicación `conf/xml/`, se ejecuta el comando

```
yarprobotinterface --config RobotPositionController_iCub.xml
```

Como resultado de esto, el dispositivo tomará control de la venta de *yarpmotorgui*, lo que le permitirá modificar la posición de los ángulos del robot.

9. **Iniciar detector de pose y estimación de ángulos:** Finalmente, en una nueva terminal abierta desde la carpeta donde se haya clonado el repositorio “`vision-system-for-robot-teleoperation`”, se ejecuta el archivo `webcam_vector_write_port_inv.py`, utilizando el comando

```
python3 webcam_vector_write_port_inv.py
```

Esto activará la cámara y dará inicio a la detección de pose del operador e inferencia de ángulos.

Al ejecutar todos estos pasos, el robot comenzará a replicar las poses y movimientos captados por la *webcam* externa. Para detener la teleoperación, se recomienda ir desactivando cada módulo en el orden contrario al que fueron activados, es decir, deteniendo cada módulo desde el paso 9 al paso 1.

5.3. Registro y análisis de resultados: subsistemas

Originalmente, los experimentos se llevarían a cabo en un entorno (contenedor) utilizando *Docker*, con el propósito de garantizar su replicabilidad. Sin embargo, después de varios experimentos fallidos (principalmente de los experimentos del sistema general) se desiste de esta opción. El motivo por el cual no fue posible realizar los experimentos utilizando este método es porque los contenedores no lograban reconocer un servidor *YARP* que se encontrara fuera de este ambiente, lo que imposibilitaba la conexión entre los sistemas de control y el robot real.

Debido a la imposibilidad de trabajar con el contenedor de *Docker*, se opta por instalar todos los repositorios, *software* y librerías necesarias de forma local, ya sean los necesarios para la operación del robot iCub como los desarrollados en este trabajo. Con esto, se garantiza la comunicación entre el robot real y el sistema desarrollado. Además, al instalar el *software* de forma local en el computador, se optimiza el uso de CPU y, por lo tanto, las inferencias y velocidad ejecución de los programas desarrollados sería mayor.

5.3.1. A: Visión

Resultados: *Pose Landmark* versus *Pose World Landmark*

En las Figuras 5.14 y 5.15 se presentan los gráficos obtenidos al medir la variación de la distancia entre los diferentes pares de articulaciones definidos mientras el operador realiza la pose “T”. Se observa claramente que al utilizar *pose_landmarks*, la proporción de las extremidades definidas exhibe deformaciones notables en el esquema generado de la postura humana. En contraste, al emplear *pose_world_landmarks*, las deformaciones son considerablemente menores, lo que sugiere una mayor coherencia y reducción de deformidades en la representación de la pose obtenida. Esta discrepancia sugiere que al utilizar las coordenadas *pose_world_landmarks*, la silueta resultante de la pose poseerá una mayor coherencia y presentará deformaciones menores.

Considerando estos hallazgos, se toma la decisión de emplear las coordenadas (x, y, z) provenientes de las *pose_world_landmarks* como la representación de la pose detectada. Estos valores se utilizarán como entrada para los subsistemas restantes, en virtud de su capacidad para proporcionar una representación más precisa y coherente de la postura detectada.

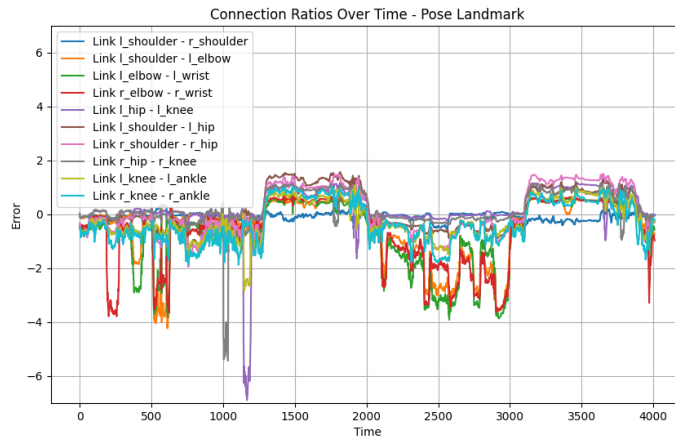


Figura 5.14: Variación de la longitud de las extremidades utilizando como inferencia *Pose Landmarks*.

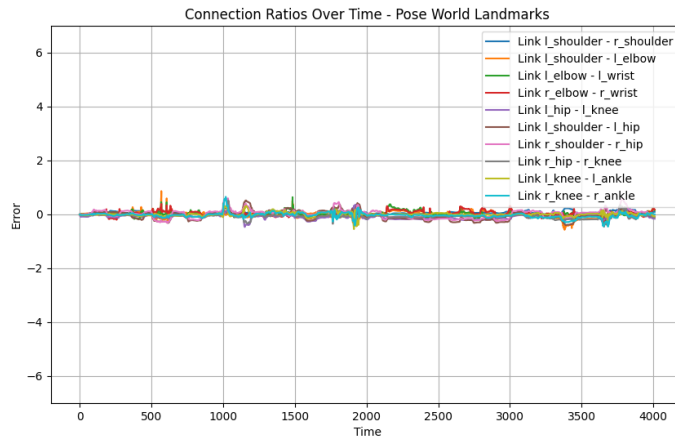


Figura 5.15: Variación de la longitud de las extremidades utilizando como inferencia *Pose World Landmarks*.

Detección de pose en tiempo real

Los tiempos registrados para los primeros 20 *frames* de un video en vivo realizando la inferencia de la posición utilizando el subsistema de visión dentro de un contenedor *Docker* y de forma local se registran en la Tabla 5.3.

De los valores registrados para las pruebas utilizando *Docker*, el mayor tiempo de inferencia corresponde a 0.2542 segundos, mientras que el menor tiempo registrado corresponde a 0.0501 segundos. En promedio, el tiempo de inferencia de un *frame* es de 0.0674 segundos. Para el caso de pruebas realizadas de forma local, el mayor tiempo de inferencia corresponde a 0.0520 segundos, mientras que el menor tiempo registrado corresponde a 0.0417 segundos y teniendo un promedio de inferencia de 0.0446 segundos. En ambos casos, el mayor tiempo de inferencia corresponde al *frame* 1, lo que se puede atribuir a la inicialización del sistema de inferencia.

Como era esperado, los tiempos de inferencia utilizando el contenedor *Docker* son mayores a los de forma local. Además, los tiempos de inferencia son mayores a los declarados por *Mediapipe* (Tabla 5.2). El aumento de inferencia se atribuye a las diferencias en el modelo del procesador de computador usado (HP-Pavilion con procesador Intel® Core™ i5-6200U).

Tabla 5.3: Tiempos de inferencia de *Mediapipe Pose* para 20 *frames*, realizando la inferencia dentro de un contenedor *Docker* y de forma local.

Nro. de Frame	Tiempo de inferencia en Docker [s]	Tiempo de inferencia local [s]
1	0.2542	0.0520
2	0.0681	0.0477
3	0.0632	0.0480
4	0.0537	0.0482
5	0.0778	0.0444
6	0.0631	0.0461
7	0.0501	0.0456
8	0.0532	0.0454
9	0.0581	0.0430
10	0.0524	0.0474
11	0.0511	0.0417
12	0.0505	0.0446
13	0.0562	0.0468
14	0.0555	0.0471
15	0.0504	0.0431
16	0.0593	0.0434
17	0.0539	0.0493
18	0.0568	0.0454
19	0.0669	0.0491
20	0.0547	0.0426
Promedio	0.06021	0.0446

5.3.2. B: Transformación de data

A continuación se registran los resultados obtenidos para los experimentos del subsistema B: Transformación de data.

Resultados: Estimación de ángulos

En la Figura 5.16 se han registrado las poses detectadas reconstruidas a partir de los ángulos estimados para 4 *frames* del video. Se puede observar que el esqueleto generado mantiene un tamaño relativamente constante en sus extremidades, sin mostrar deformaciones significativas. Las poses generadas muestran una gran similitud visual con la pose original deseada.

Una consideración importante radica en que, al estimar los ángulos para representar las extremidades, no se han considerado las limitaciones inherentes al movimiento del cuerpo humano. Por ejemplo, en el caso del brazo humano, la articulación del codo solo puede extenderse hasta los 180° , ya que cualquier ángulo mayor resultaría en una fractura del brazo del individuo. Sin embargo, en el gráfico generado, la articulación del codo tiene una capacidad de movimiento de 360° , sin restricciones. Asimismo, algunos ángulos de rotación en las articulaciones pueden ser difíciles de detectar utilizando únicamente visión computacional. Por ejemplo, al considerar las rotaciones en torno a su propio eje de un brazo está extendido, el algoritmo de estimación de postura solo podría detectar la extensión del brazo, lo que conlleva a un posible error en la estimación de la postura y, por ende, a posiciones inexactas al replicar dicha postura en un robot.

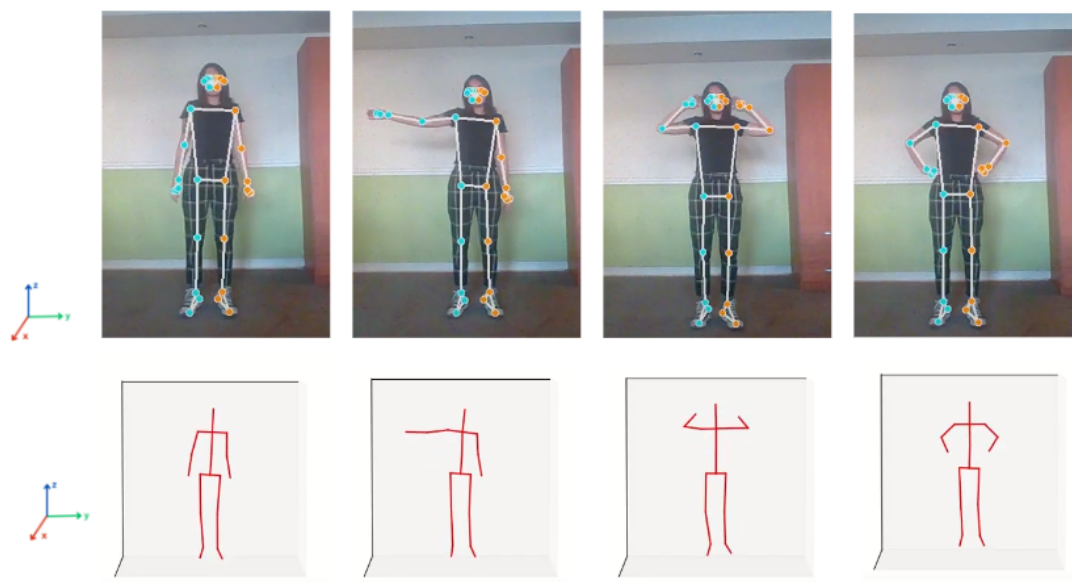


Figura 5.16: Reconstrucción de pose detectada utilizando ángulos estimados por el subsistema B.

Resultados: Comunicación mediante red YARP

Uno de los desafíos del desarrollo de esta solución era asegurar la correcta comunicación entre subsistemas escritos en diferentes lenguajes de programación (*C++* y *Python*). El resultado obtenido del chequeo de la comunicación entre el puerto de lectura del servidor *YARP* y el puerto de escritura encargado de enviar los vectores de datos definidos en *Python* se muestran en las Figuras

5.17 y 5.18. En la Figura 5.17 se destaca el mensaje informativo de la terminal indicando que el puerto de escritura (nombrado */wport*) para el envío de datos lectura se ha creado exitosamente, y se logra conectar sin problemas a su puerto cliente (en este caso, */dataPort*). En la Figura 5.18 se muestra el funcionamiento completo del subsistema de transformación de datos, incluyendo la terminal con el servidor YARP funcionando, el puerto de lectura de vector de datos con valores ya recibidos, el puerto con el subsistema B funcionando, la imagen de entrada del sistema y finalmente el gráfico generado para la pose utilizando los ángulos estimados.

```

Terminal - root@725c7099632b: ~/repos/icub_vision/real_tim ^ _ □ ×
File Edit View Terminal Tabs Help
root@725c7099632b:~/repos/icub_vision/real_time# py
thon3 webcam_vector_write_port_
webcam_vector_write_port_2.py
webcam_vector_write_port_hc.py
root@725c7099632b:~/repos/icub_vision/real_time# py
thon3 webcam_vector_write_port_2.py
INFO: Created TensorFlow Lite XNNPACK delegate for
CPU.
[INFO] |yarp.os.Port|/wport| Port /wport active at
tcp://172.17.0.2:10003/
[INFO] |yarp.os.impl.PortCoreOutputUnit|/wport| Sen
ding output from /wport to /dataPort using tcp
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting
to '/tmp/runtime-root'
webcam_vector_write_port_2.py:568: DeprecationWarni
ng: The binary mode of fromstring is deprecated, as
it behaves surprisingly on unicode inputs. Use fro
mbuffer instead
    plot_img = np.fromstring(fig.canvas.tostring_rgb(
), dtype=np.uint8,
[INFO] |yarp.os.impl.PortCoreOutputUnit|/wport| Rem
oving output from /wport to /dataPort
^CTraceback (most recent call last):
  File "webcam_vector_write_port_2.py", line 581, i
n <module>
    time.sleep(0.08)
KeyboardInterrupt

```

Figura 5.17: Envío y recepción correcta de paquetes de datos desde el subsistema B a un puerto de lectura YARP.

Resultados: Tiempo de generación de datos

Finalmente, en la Tabla 5.4 se registra el tiempo de inferencia, generación y envío de vector de datos para los primero *20 frames* de video. El mayor tiempo registrado fue de 0.8143 segundos, el menor tiempo fue de 0.0717 segundos y en promedio el tiempo de desfase entre la detección de la pose y el envío del vector de datos es de 0.1366 segundos. Este desfase es mayor al registrado para el subsistema A, y se atribuye al tiempo de cálculo de las estimaciones de ángulo y velocidad de los ángulos (sea o no el caso). En cuanto a la experiencia de usuario, este desfase entre el movimiento y el tiempo de cómputo es levemente perceptible por el operador sobre todo cuando se realizan movimientos de forma rápida y fluida.

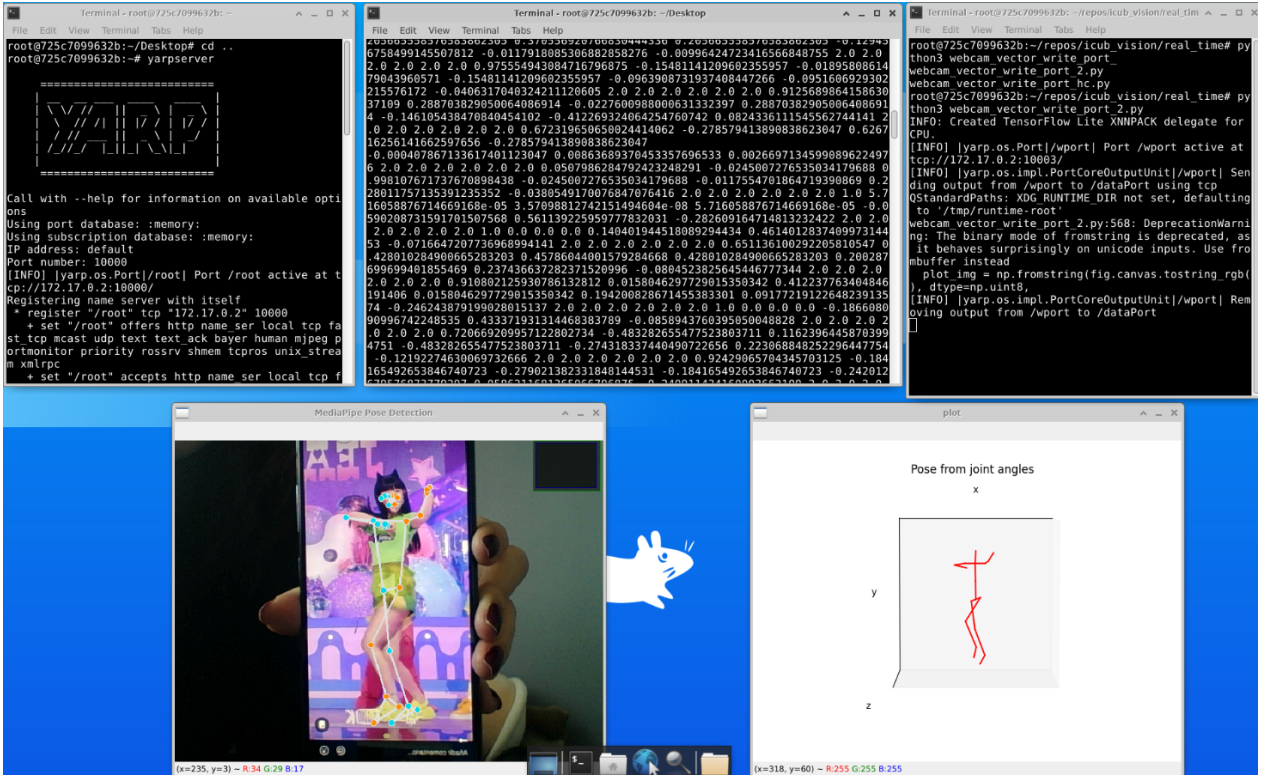


Figura 5.18: Conexión exitosa entre puerto de escritura creado usando *Python* y puerto de lectura definido usando comandos de *YARP*.

Tabla 5.4: Tiempos de inferencia y envío de vector de datos para 20 frames (subsistema B).

Nro. de Frame	Tiempo de inferencia [s]
1	0.8143
2	0.1391
3	0.0740
4	0.1164
5	0.0754
6	0.0764
7	0.0738
8	0.0810
9	0.0744
10	0.1130
11	0.0755
12	0.1135
13	0.1116
14	0.1096
15	0.1859
16	0.1093
17	0.1088
18	0.0717
19	0.0848
20	0.1250

5.3.3. C: Control

Ejecución de pose objetivo

Para este experimento se utilizaron vectores de datos configurados manualmente para chequear que la lectura de los datos se realiza correctamente y que la información de posición y velocidad de cada *joint* es relacionada correctamente a su actuador objetivo.

En la Figura 5.19, se muestran los resultados obtenidos utilizando la visualización y el simulador Gazebo para un vector de datos compuesto de valores nulos excepto para la articulación del hombro derecho del robot. EL vector de datos usado y las instrucciones para replicar este experimento se encuentran en el repositorio *human-dynamics-estimation-with-vision*.

De las imágenes obtenidas se observa que el vector que describe la pose objetivo de entrada es correctamente leído y se mapea correctamente la pose al cuerpo del robot, reemplazando completamente la información generada por los sensores de movimientos que era originalmente utilizados en este proyecto. El tiempo de respuesta del controlador es casi inmediato en el visualizador, pero en el entorno de simulación de *Gazebo* se aprecia una leve demora, lo que puede ser a causa de las capacidades de computo del computador utilizado. Para saber si la demora en el cambio de pose es debido al entorno de simulación o es una limitación del robot, se hará la comparación entre el tiempo de reacción con el robot real.

La diferencia de tiempo entre una pose y otra también estará determinado por la frecuencia del envío de los vectores de datos desde el subsistema C.

En resumen, se logra re definir el controlador propuesto en el repositorio *human-dynamics-estimation*, quitando su dependencia de hardware extra como los sensores de captura de movimiento.

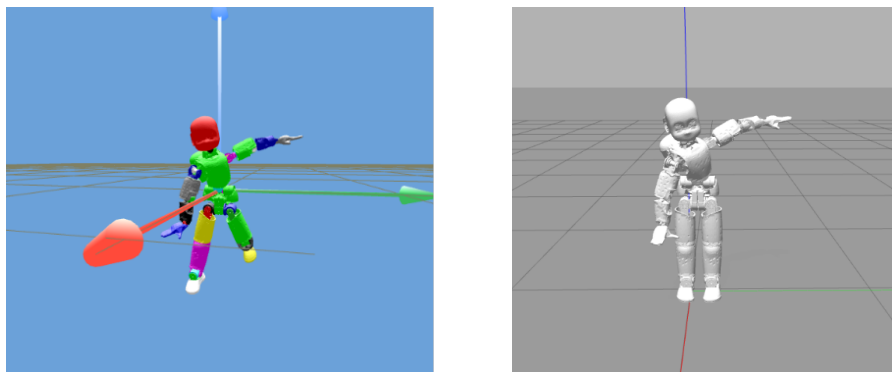


Figura 5.19: Visualización (izquierda) y simulación (derecha) de los movimientos del robot iCub dada una pose objetivo.

5.4. Registro y análisis de resultados: sistema general

Ya realizadas las pruebas en los diferentes subsistemas que componen la solución, se procede a integrar todo en un sistema único y se realizan las pruebas pertinentes utilizando el simulador *Gazebo* y en el robot real. A continuación se registran los resultados.

5.4.1. Control en simulación

Los resultados cualitativos obtenidos para la teleoperación de la simulación se presentan en la Figura 5.20. Para este caso, el sistema completo fue ejecutando en un computador (simulación de teleoperación local). De la imagen se puede apreciar que la simulación es capaz de replicar la posición capturada por el sistema de visión, pero respetando sus límites físicos. Particularmente, de el segundo *frame* mostrado en la imagen, se observa que el actuador del hombro derecho del iCub simulado no le permite alcanzar la misma posición que el operador, dejando su brazo derecho levemente separado de su torso, a diferencia del operador que lo mantiene junto al cuerpo.

Cuando la pose a imitar excede alguno de los límites de las articulaciones de la simulación o representaba una pose peligrosa para el *hardware* del robot, el controlador definido (*Human State Provider*) deja de operar de manera inmediata y desconecta la simulación de la información de entrada. En tales casos, la simulación quedaba posicionada en la última pose segura detectada.

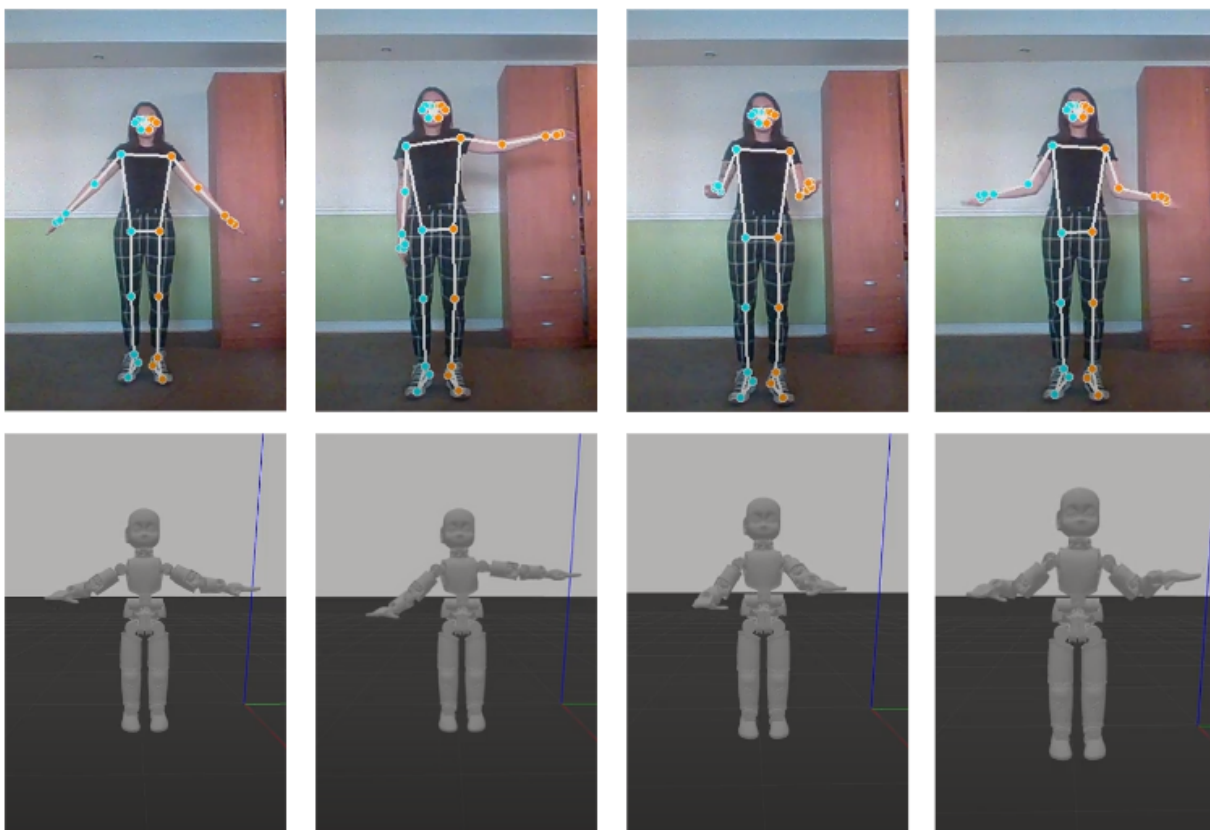


Figura 5.20: Teleoperación de simulación de robot iCub v2.5 en tiempo real. El robot se encuentra fijo desde su cadera sin tener contacto con el suelo (superficie flotante).

Para evaluar de forma cuantitativa los resultados obtenidos, se obtienen los vectores que contienen los ángulos de entrada al controlador, correspondientes a los estimados de las articulaciones del operador, y de los ángulos de salida, correspondientes a los ángulos simulados alcanzados por el robot iCub en simulación. Se realizan 10 experimentos por cada extremidad, y se almacenan ambos vectores en un archivo de texto junto a su *timestamp* correspondiente.

Para medir la diferencia entre la posición de entrada y salida del sistema, se calcula el error cuadrático medio (*root mean square error* o RMSE por sus siglas en inglés) entre la entrada y salida del sistema. Se decide utilizar esta métrica de evaluación ya que permite medir la diferencia promedio entre los valores estimados de entrada y los valores de salida en la misma escala que los datos originales. Además, esta métrica tiene una alta sensibilidad a errores grandes debido al término de cuadrado en su cálculo por lo que los errores grandes tendrán un impacto significativo en el valor del RMSE, lo que lo hace más sensible a los errores importantes. Esta característica del RMSE resulta útil en este caso, ya que es importante lograr identificar si existen grandes diferencias entre la pose de entrada y salida.

El experimento se realiza diez veces para las articulaciones, se registran los resultados en una tabla obteniendo el error promedio y se generan gráficos comparativos del ángulo de entrada y el ángulo alcanzado por el actuador del robot en simulación en grados (si bien los ángulos son trabajados en *quaterniones* por el sistema, se grafican en grados para facilitar su lectura y comparación). En los gráficos, el eje izquierdo de ángulos corresponde al eje para la curva del ángulo de entrada (en inglés *input angle*) y el de la derecha corresponde al eje para la curva del ángulo del robot (en inglés *robot angle*). La diferencia de eje entre las curvas corresponde a un factor de escala.

Los valores obtenidos para las *caderas izquierda y derecha* se registran en la Tabla 5.5, y sus gráficos comparativos se ilustran en las Figuras 5.21 y 5.22. Los valores obtenidos para las *rodillas izquierda y derecha* se registran en la Tabla 5.6, y sus gráficos comparativos se ilustran en las Figuras 5.23 y 5.24. Para los *talones izquierdo y derecho* se registran los resultados en la Tabla 5.7, y sus gráficos comparativos en las Figuras 5.25 y 5.26.

Para las extremidades superiores, los valores obtenidos para los *hombros izquierdo y derecho* se registran en la Tabla 5.8, y sus gráficos comparativos se ilustran en las Figuras 5.27 y 5.28; y para los *codos izquierdo y derecho* se registran en la Tabla 5.9, y sus gráficos comparativos se ilustran en las Figuras 5.29 y 5.30.

Cabe destacar que no en todos los casos estarán presentes la comparación de los ángulos *roll*, *pitch* y *yaw*, ya que el robot real no posee estos tres ejes de movimiento en todas las extremidades. En estos casos, se muestra solamente la comparación entre los ángulos análogos.

Tabla 5.5: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para las caderas.

Experimentos	Cadera izquierda			Cadera derecha		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	0.4533	62.9327	0.3575	0.6962	2.418	1.7237
Exp. 2	0.4190	3.4406	0.8202	0.6225	5.3804	0.5741
Exp. 3	0.3095	181.4567	0.4579	0.7544	3.1248	2.8601
Exp. 4	0.5336	2.7752	0.4740	0.8101	2.0123	2.8914
Exp. 5	0.3544	5.8827	0.2947	0.5651	1.6231	0.7825
Exp. 6	0.2581	8.2008	0.2234	0.7456	2.8451	0.8505
Exp. 7	0.3858	5.3164	0.3438	0.5933	2.1151	1.0289
Exp. 8	0.3760	1.6578	0.3548	-	-	-
Exp. 9	0.3439	4.8204	0.4185	0.5450	2.8486	0.6216
Exp. 10	0.3078	3.2061	0.3559	0.5297	2.5479	0.4636
Promedio	0.3744	22.0679	0.3870	0.6481	2.6006	1.2409

Tabla 5.6: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para las rodillas.

Experimentos	Rodilla izquierda			Rodilla derecha		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	-	7185.1305	-	-	4797.9274	-
Exp. 2	-	6923.6173	-	-	5983.3331	-
Exp. 3	-	4979.5172	-	-	8522.9506	-
Exp. 4	-	4640.1085	-	-	3333.3867	-
Exp. 5	-	26474.2943	-	-	3587.6679	-
Exp. 6	-	6624.1829	-	-	5826.5827	-
Exp. 7	-	6091.5745	-	-	5354.405	-
Exp. 8	-	3770.6583	-	-	-	-
Exp. 9	-	7090.9803	-	-	4295.2705	-
Exp. 10	-	5170.1943	-	-	6408.2271	-
Promedio	-	7982.1256	-	-	4880.2319	-

Tabla 5.7: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para los talones.

Experimentos	Tobillo izquierdo			Tobillo derecho		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	0.8321	1.3410	-	8.3864	1.8866	-
Exp. 2	1.2770	0.9605	-	19.0057	2.2502	-
Exp. 3	1.8874	1.7632	-	7.6684	1.4989	-
Exp. 4	2.1163	0.9054	-	10.6119	0.8137	-
Exp. 5	0.7043	1.5141	-	2.3216	3.4055	-
Exp. 6	0.9457	1.0500	-	1.5874	4.9998	-
Exp. 7	0.8071	1.6965	-	4.3927	1.6763	-
Exp. 8	1.9852	1.5652	-	-	-	-
Exp. 9	10492.2397	1.1911	-	2.6901	2.1742	-
Exp. 10	1.0252	1.1040	-	2.0796	1.4782	-
Promedio	1050.0869	1.3086	-	7.5628	2.2192	

Tabla 5.8: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para los hombros.

Experimentos	Hombro izquierdo			Hombro derecho		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	7.6880	0.9322	1.5257	1.3150	6.9284	0.8230
Exp. 2	7.5276	0.4033	3.9129	1.3435	1.9589	0.8864
Exp. 3	7.0116	0.8899	2.2357	1.3996	1.5218	0.8974
Exp. 4	5.2586	0.6626	3.8277	1.3762	1.9505	1.1118
Exp. 5	3.8369	0.6089	5.0884	1.2869	1.3509	0.8662
Exp. 6	12.1236	2.1571	4.9208	1.3073	6.0714	0.8004
Exp. 7	15.2210	0.5257	1.8751	1.3556	1.3562	1.1706
Exp. 8	30.7071	0.2844	3.7748	1.3672	0.9788	1.4511
Exp. 9	4.3631	1.6275	6.5377	1.4721	2.5269	1.0048
Exp. 10	7.4651	2.0501	4.2167	1.3983	1.6446	1.6067
Promedio	10.1189	0.9132	3.7915	1.3021	2.9288	0.9618

Tabla 5.9: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot (simulación) para los codos.

Experimentos	Codo izquierdo			Codo derecho		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	-	0.4142	-	-	0.4973	-
Exp. 2	-	0.2026	-	-	0.4155	-
Exp. 3	-	0.2525	-	-	0.3225	-
Exp. 4	-	0.2145	-	-	0.3119	-
Exp. 5	-	0.3075	-	-	0.4251	-
Exp. 6	-	0.3520	-	-	0.4131	-
Exp. 7	-	0.2939	-	-	0.3684	-
Exp. 8	-	0.3975	-	-	0.4844	-
Exp. 9	-	0.2653	-	-	0.4169	-
Exp. 10	-	0.1806	-	-	0.4140	-
Promedio	-	0.2880	-	-	0.4168	-

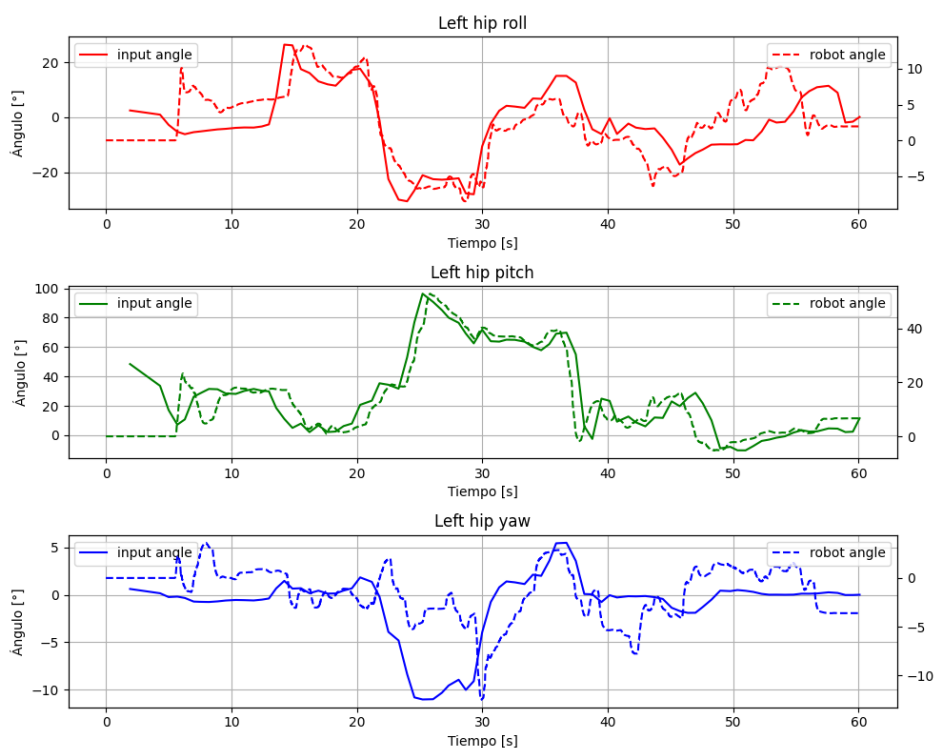


Figura 5.21: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para la cadera izquierda.

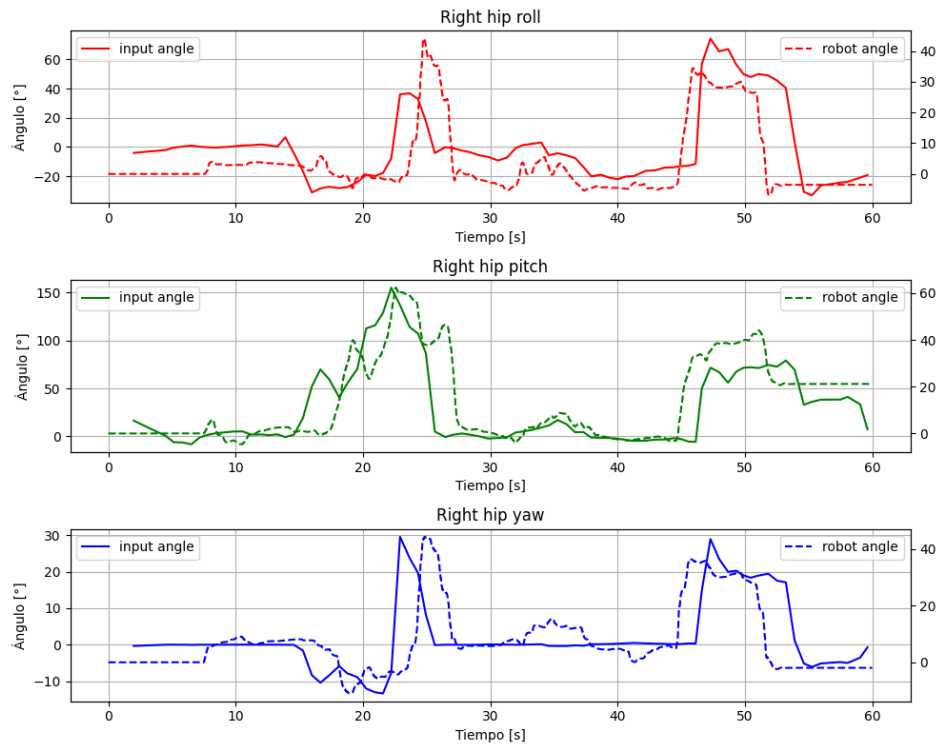


Figura 5.22: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para la cadera derecha.

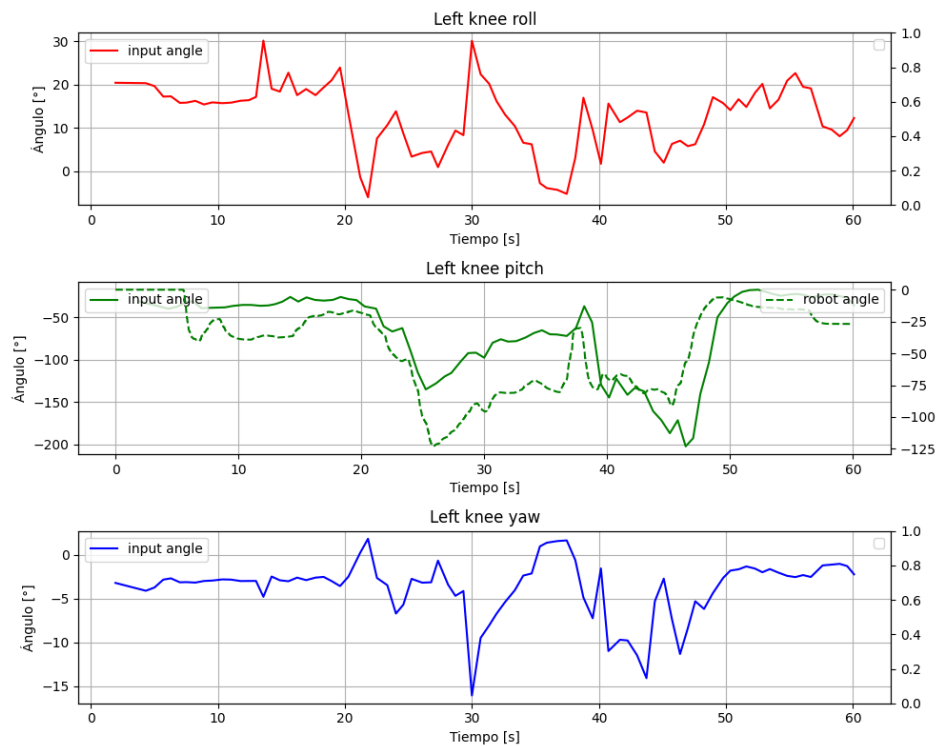


Figura 5.23: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para la rodilla izquierda.

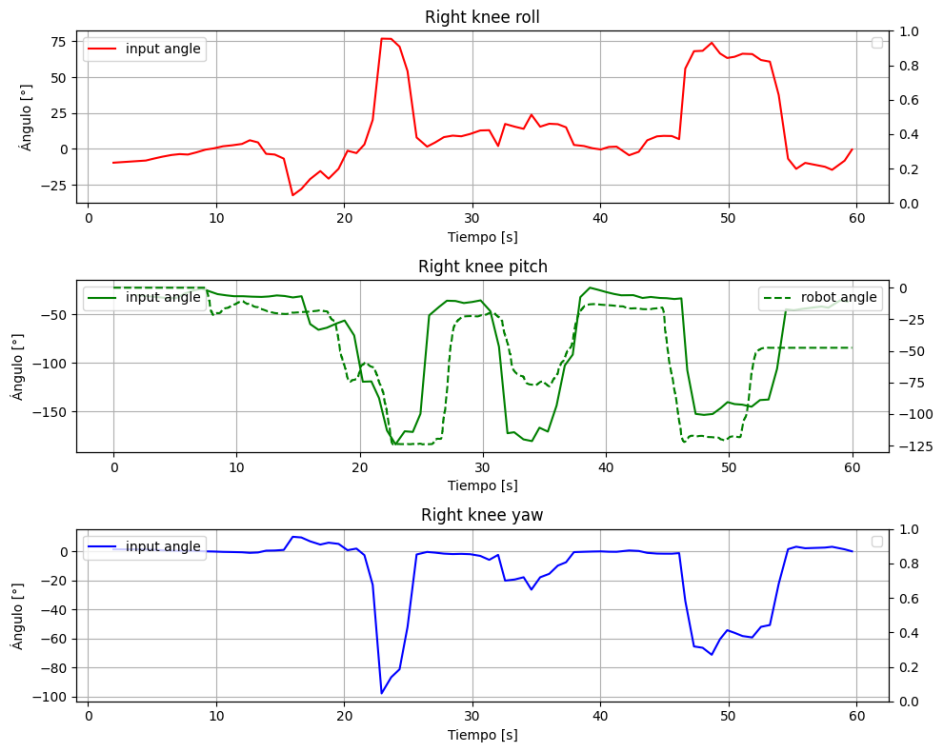


Figura 5.24: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para la rodilla derecha.

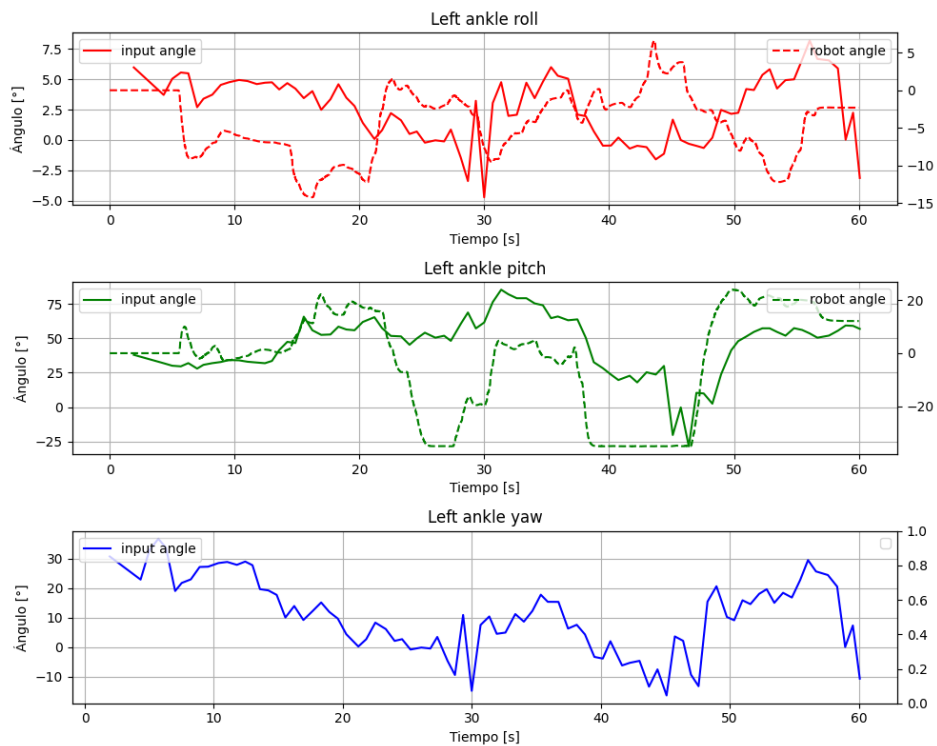


Figura 5.25: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el tobillo izquierdo.

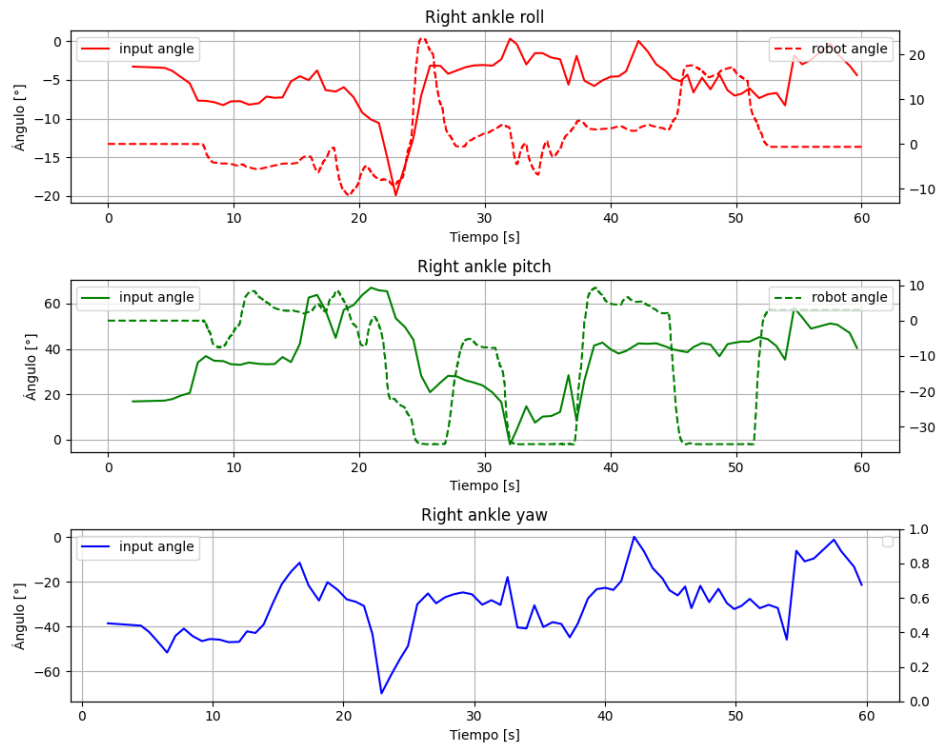


Figura 5.26: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el tobillo derecho.

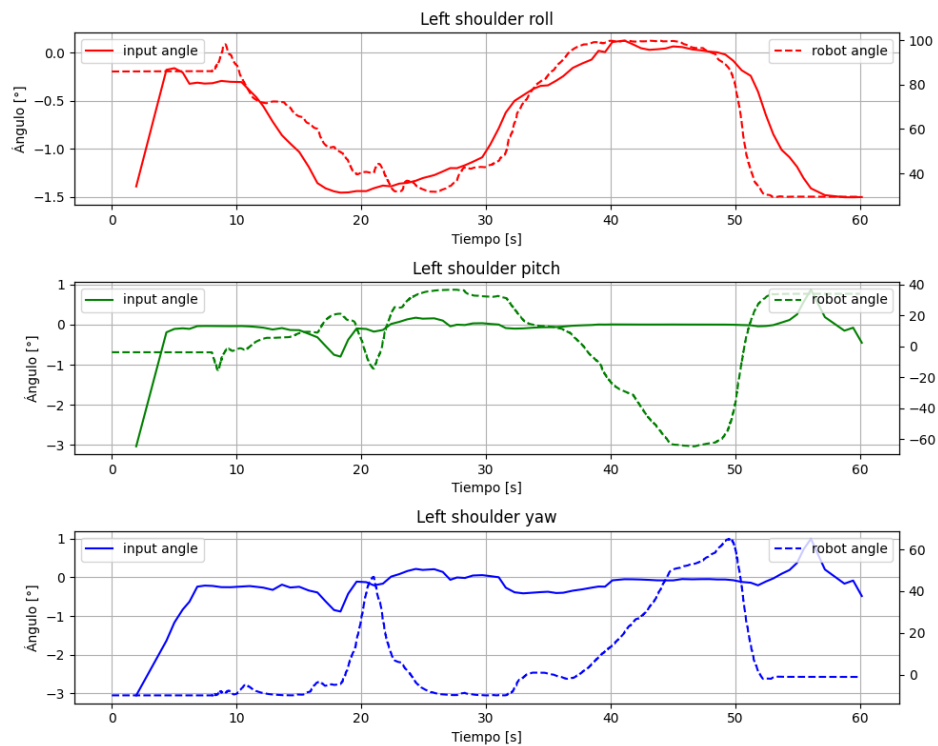


Figura 5.27: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el hombro izquierdo.

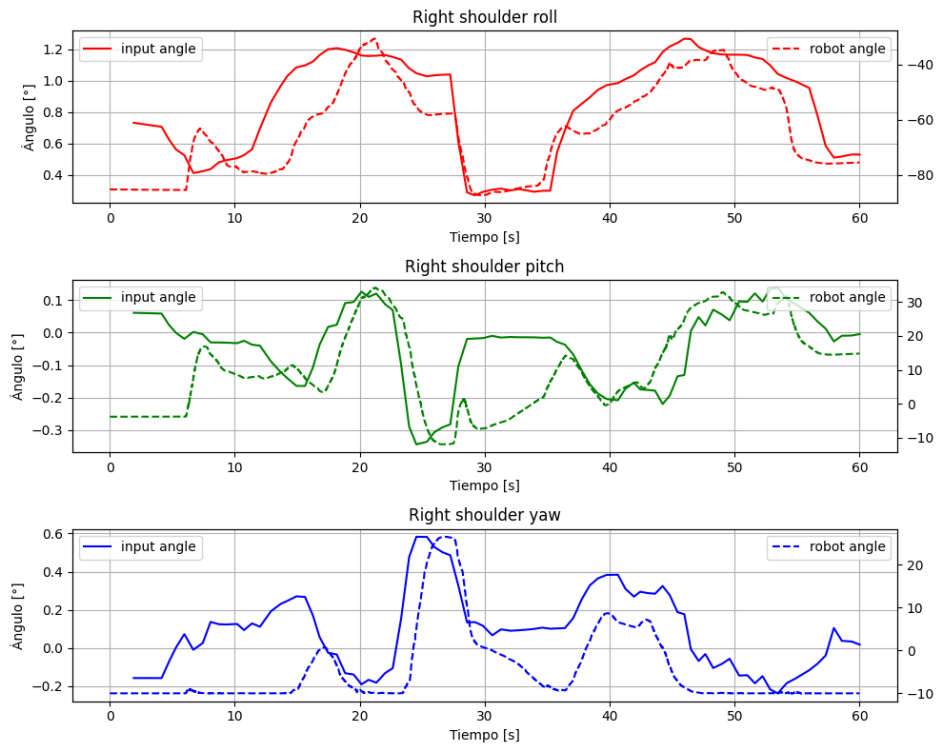


Figura 5.28: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el hombro derecho.

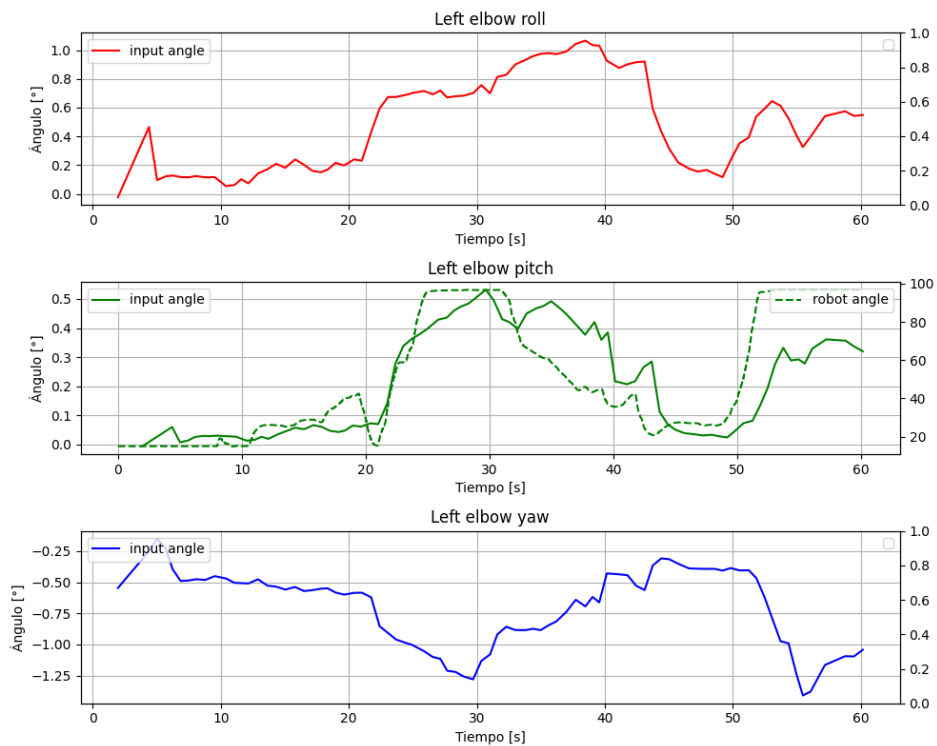


Figura 5.29: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el codo izquierdo.

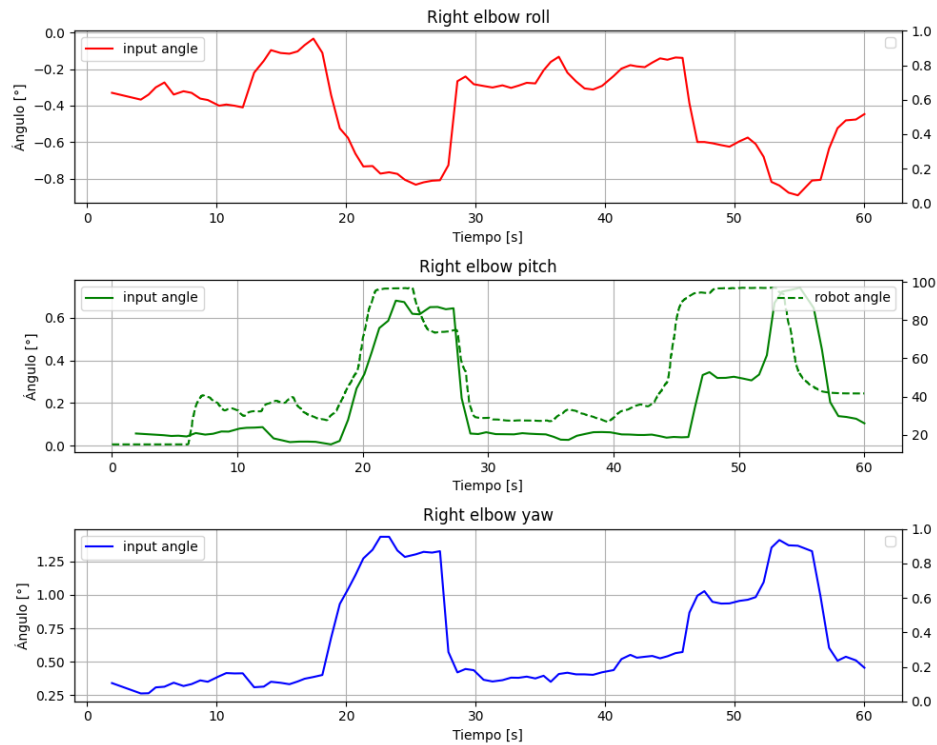


Figura 5.30: Comparación entre ángulos de entrada y ángulo alcanzado por el robot (simulación) para el codo derecho.

5.4.2. Control en robot real

Para evaluar el funcionamiento del sistema completo utilizando el robot real, se realizan los mismos experimentos realizados para el caso de *control en simulación*, pero con 5 repeticiones por articulación. A continuación se detallan los resultados obtenidos.

Los resultados cualitativos obtenidos para la teleoperación del robot real se muestran en la Figura 5.31, ejecutando el sistema completos desde un solo computador (sistema de teleoperación local). De la imagen se puede observar que el robot es capaz de imitar la posición capturada por el sistema de visión, pero respetando sus límites físicos (lo que genera diferencias con la pose objetivo). Al igual que en el caso de simulación, cuando la pose a replicar superaba alguno de los límites de las articulaciones del robot, el controlador definido (*Human State Provider*) deja de funcionar de forma inmediata y desconecta al robot de la información de entrada, protegiendo la estructura del robot. En estos casos, el robot queda posicionado en la última pose segura detectada.

En cuanto al tiempo transcurrido desde que se realiza la pose hasta que el robot la imita, este varía entre 1 y 2 segundos. El tiempo que le tarde al robot alcanzar la pose no solo depende de la velocidad de inferencia y del sistema de control, sino que también depende de el desplazamiento que deban realizar sus articulaciones: si la pose a imitar difiere en gran medida de la pose anterior realizada, le tardará unos instantes al robot mover todas sus extremidades para alcanzar la nueva pose requerida.

Los experimentos realizados fueron registrados en imágenes y video. Estos se encuentran dis-

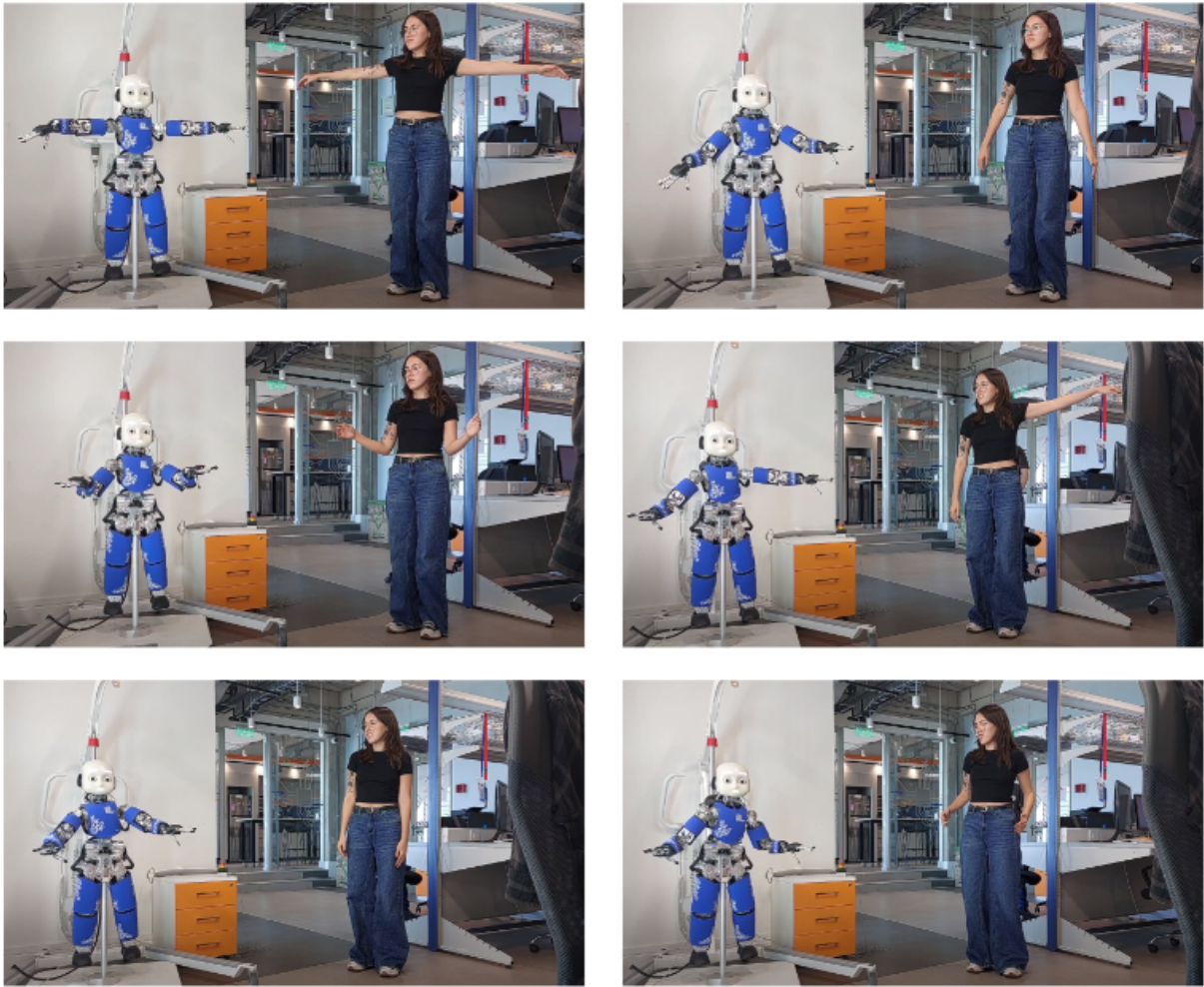


Figura 5.31: Teleoperación de robot iCub v2.7 en tiempo real. El robot se encuentra fijo desde su cadera sin tener contacto con el suelo (superficie flotante).

ponibles en el repositorio de *GitHub iCub-vision-teleoperation*².

Comparación: ángulos de entrada y *encoders* en el robot real

Para evaluar numéricamente los resultados obtenidos, al igual que para el caso en simulación, se generan gráficos comparativos del ángulo de entrada y el ángulo real alcanzado por el actuador del robot para cada articulación, y se registra la *raíz del error cuadrático medio* en tablas.

Los valores obtenidos para las *caderas izquierda y derecha* se registran en la Tabla 5.10, y sus gráficos comparativos se ilustran en las Figuras 5.32 y 5.33. Los valores obtenidos para las *rodillas izquierda y derecha* se registran en la Tabla 5.11, y sus gráficos comparativos se ilustran en las Figuras 5.34 y 5.35. Para los *talones izquierdo y derecho* se registran los resultados en la Tabla 5.12, y sus gráficos comparativos en las Figuras 5.36 y 5.37.

Para las extremidades superiores, los valores obtenidos para los *hombros izquierdo y derecho* se

²Disponible en <https://github.com/f4lfaro/iCub-vision-teleoperation>.

registran en la Tabla 5.13, y sus gráficos comparativos se ilustran en las Figuras 5.38 y 5.39; y para los *codos izquierdo y derecho* se registran en la Tabla 5.14, y sus gráficos comparativos se ilustran en las Figuras 5.40 y 5.41.

De los experimentos realizados, las Figuras 5.32 y 5.33 correspondiente a las caderas son las que presentan una mayor exactitud en respecto al movimiento del humano y del robot. Para estos casos, las curvas de los ángulos *roll*, *pitch* y *yaw* coinciden con los movimientos del robot. El desfase apreciado se debe al tiempo de cálculo y *retargeting* de la pose objetivo. Estos resultados coinciden con los valores registrados en la Tabla 5.10, ya que esta presenta el menor grado de error comparado al resto de las extremidades. Parte de esta gran exactitud se atribuye a que los tres ángulos estimados *roll*, *pitch* y *yaw* si se encuentran presentes en el robot real y, por lo tanto, la imitación de la pose es directa.

En el caso de las articulaciones rodilla (Figuras 5.34 y 5.35) y talón (Figuras 5.36 y 5.37), no todos los ángulos estimados de la pose del humano se encuentran presentes en el rango de movimiento del robot. Sin embargo, los ángulos son enviados al controlador en forma de *quaterniones*, por lo que este logra realizar una estimación del movimiento deseado considerando las limitaciones del robot.

Para el caso de la rodilla, la Tabla 5.11 muestra un error de gran magnitud, sin embargo, los resultados registrados en el gráfico de las Figuras 5.34 y 5.35, y los movimientos apreciados en el video de los experimentos no muestran grandes diferencias cualitativas entre los movimientos del operador y del robot, por lo que la diferencia se atribuye a la diferencia de grados de libertad estimados del operador (ángulos *roll*, *pitch* y *yaw* para la rodilla) y los reales presentes en el robot (ángulo *pitch* en la rodilla).

Para el caso de los brazos, en algunos instantes de tiempo la pose real alcanzada por el robot presenta diferencias de la pose objetivo. Las discrepancias más notorias se observan para el hombro izquierdo (Figura 5.38), donde las mediciones de los tres ángulos muestran un comportamiento diferente al esperado. En la Tabla 5.13 se registra el mismo comportamiento, mostrando el hombro izquierdo un error mayor que el hombro derecho. Sin embargo, los valores de error calculados son pequeños en comparación al de otras articulaciones. Cualitativamente, si bien las poses logradas por los brazos del robot presentan leves discrepancias, estas no coinciden con lo indicado por la Figura 5.38 pero si con lo mostrado en la Tabla 5.13, por lo que la gran discrepancia se atribuye a error de medición en el proceso experimental. En la Figura 5.39, se puede ver que la curva de la posición alcanzada por el hombro derecho del robot mantiene una forma similar a la de la pose esperada, con algunas variaciones. Estos errores se atribuyen a la aproximación de los ángulos *roll*, *pitch* y *yaw* de la imagen del humano.

En las Figuras 5.40 y 5.41, al igual que para el caso de las rodillas, no todos los ángulos estimados se encuentran presentes en el rango de movimiento de la articulación del robot. La curva de la posición alcanzada representa la aproximación a la pose objetivo realizada por el controlador. La comparación entre las curvas del robot real y del ángulo de entrada muestran discrepancias importantes, sin embargo, la Tabla 5.14 muestra que el error es menor al indicado por el gráfico. Los resultados de la tabla se ajustan más a los resultados cualitativos obtenidos y registrados en video e imágenes.

Al momento de realizar las pruebas en el robot real, ambas muñecas del robot presentaban fallas,

por los que los gráficos de comparación para esta articulación no se incluyen.

En general, los movimientos del operador logran ser replicados con similitud por el robot iCub en tiempo real.

Tabla 5.10: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para las caderas.

Experimentos	Cadera izquierda			Cadera derecha		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	0.3013	0.4134	0.2359	0.8357	0.3576	0.3926
Exp. 2	0.2134	20.4906	0.1998	0.4181	21.4580	0.5249
Exp. 3	0.3158	0.3090	0.2629	0.7129	0.7949	0.4820
Exp. 4	0.8279	12.2243	0.4995	0.6203	7.1147	0.4229
Exp. 5	0.2400	0.2400	0.2484	0.7828	2.2314	0.3728
Promedio	0.3796	6.13546	0.2893	0.6737	6.3913	0.4390

Tabla 5.11: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para los talones.

Experimentos	Rodilla izquierda			Rodilla derecha		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	-	-	-	-	0	-
Exp. 2	-	336.4176	-	-	342.4246	-
Exp. 3	-	705.0284	-	-	718.8059	-
Exp. 4	-	712.2794	-	-	0	-
Exp. 5	-	706.4932	-	-	337.4699	-
Promedio	-	615.0546	-	-	349.6751	-

Tabla 5.12: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para los talones.

Experimentos	Talón izquierdo			Talón derecho		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	0.6769	13.5975	-	3.1161	7.6608	-
Exp. 2	0.4544	2.3983	-	3.9626	9.3330	-
Exp. 3	0.6308	2.0455	-	0.3599	1.9256	-
Exp. 4	0.6785	1.897	-	4.2054	1.0810	-
Exp. 5	0.6094	1.5959	-	1.9295	0.9512	-
Promedio	0.610	4.7062	-	3.1147	4.7903	-

Tabla 5.13: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para los hombros.

Experimentos	Hombro izquierdo			Hombro derecho		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	2.0715	2.1399	6.9351	0.9322	0.7317	5.3181
Exp. 2	2.5814	1.3246	8.7092	0.8147	1.3180	2.4108
Exp. 3	5.8290	2.1348	1.6706	1.4317	0.3558	3.3650
Exp. 4	2.9204	0.7150	1.7158	1.4072	0.5867	2.9975
Exp. 5	1.0663	0.4509	4.2380	1.4797	0.8587	2.3261
Promedio	2.69372	1.1530	4.2537	1.2131	0.7702	3.2835

Tabla 5.14: Raíz del Error Cuadrático Medio (RMSE) entre los ángulos de input y output del robot real para los codos.

Experimentos	Codo izquierdo			Codo derecho		
	RMSE Roll	RMSE Pitch	RMSE Yaw	RMSE Roll	RMSE Pitch	RMSE Yaw
Exp. 1	-	0.5342	-	-	0.2985	-
Exp. 2	-	0.3526	-	-	0.5051	-
Exp. 3	-	0.5220	-	-	0.5434	-
Exp. 4	-	0.4014	-	-	0.5625	-
Exp. 5	-	0.4790	-	-	0.7187	-
Promedio	-	0.4578	-	-	0.5256	-

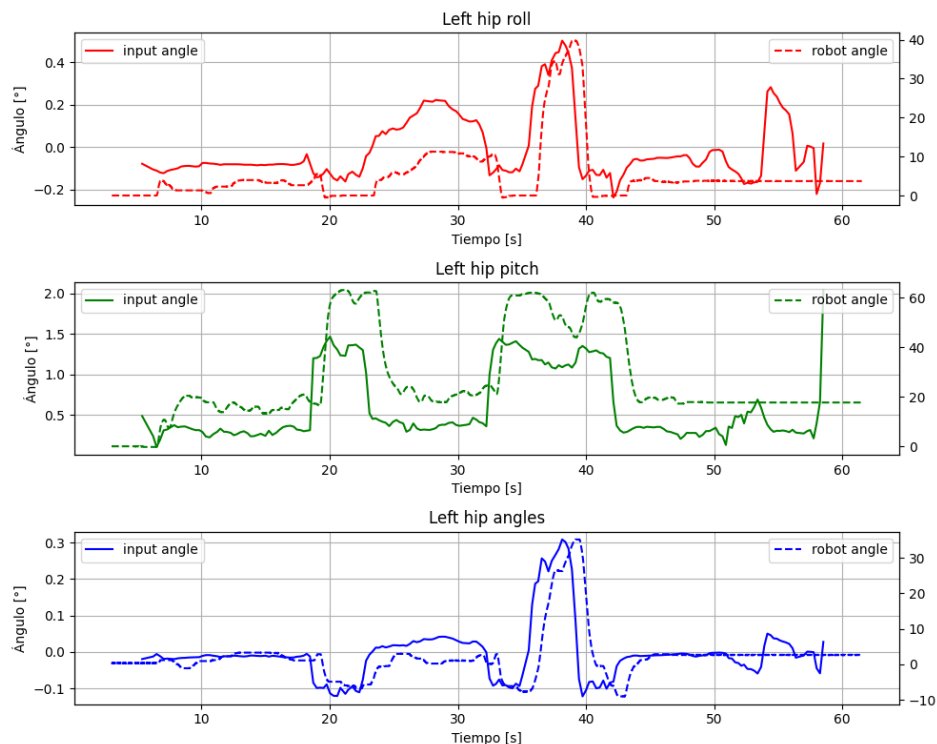


Figura 5.32: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para la cadera izquierda.

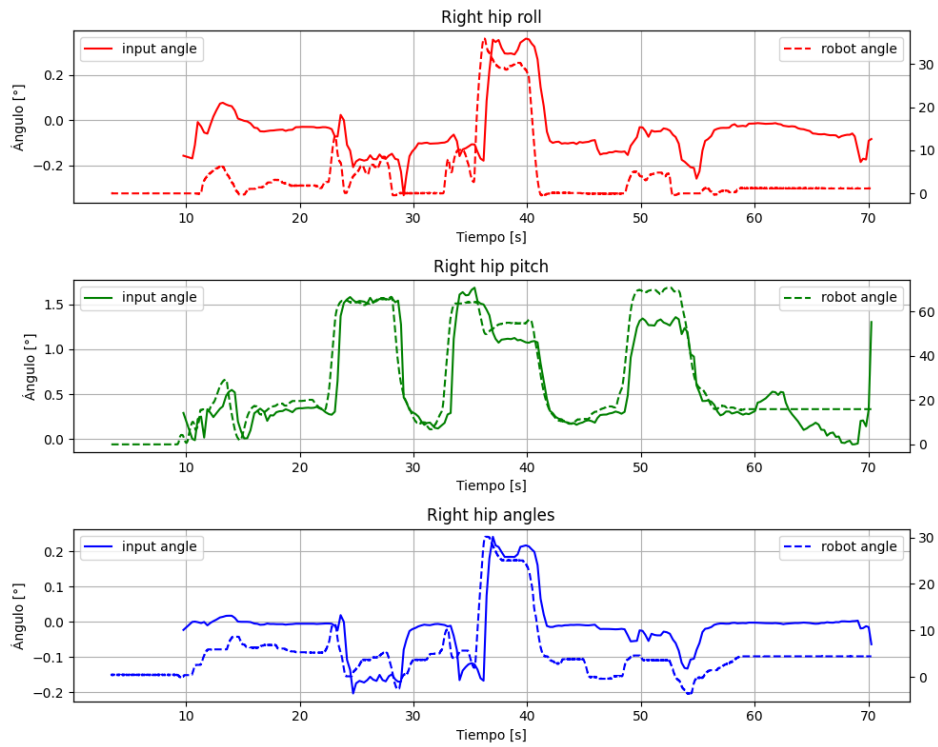


Figura 5.33: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para la cadera derecha.

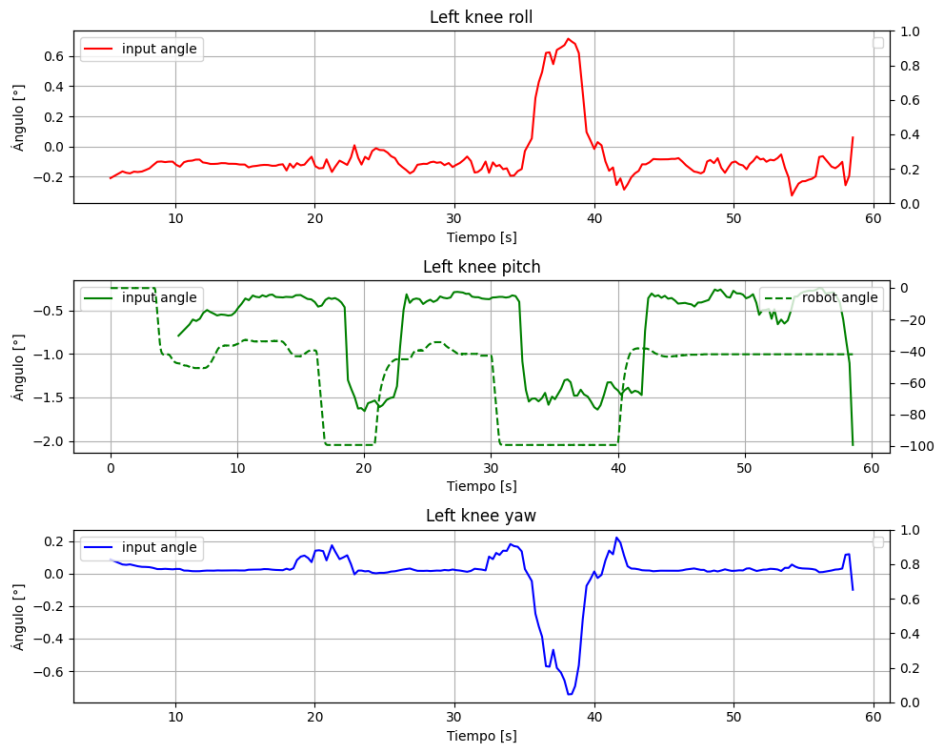


Figura 5.34: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para la rodilla izquierda.

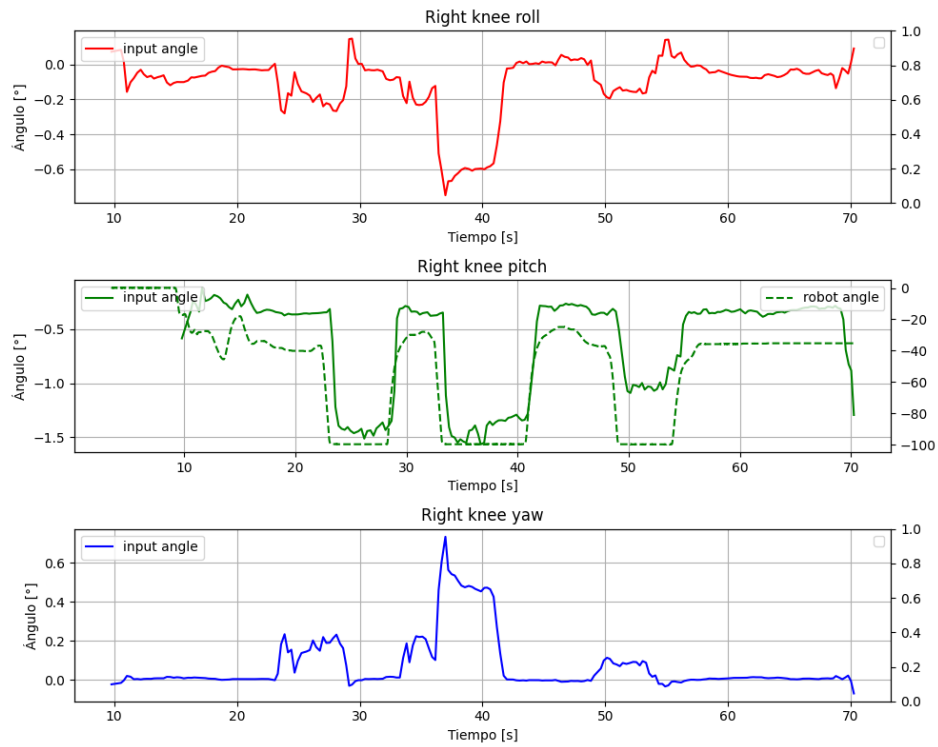


Figura 5.35: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para la rodilla derecha.

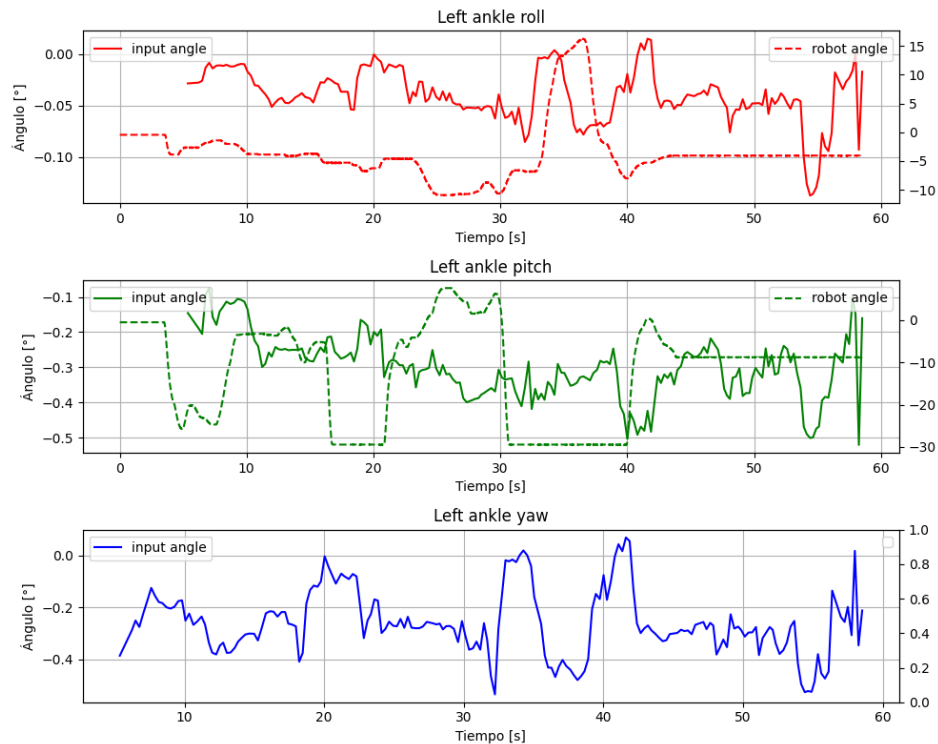


Figura 5.36: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el tobillo izquierdo.

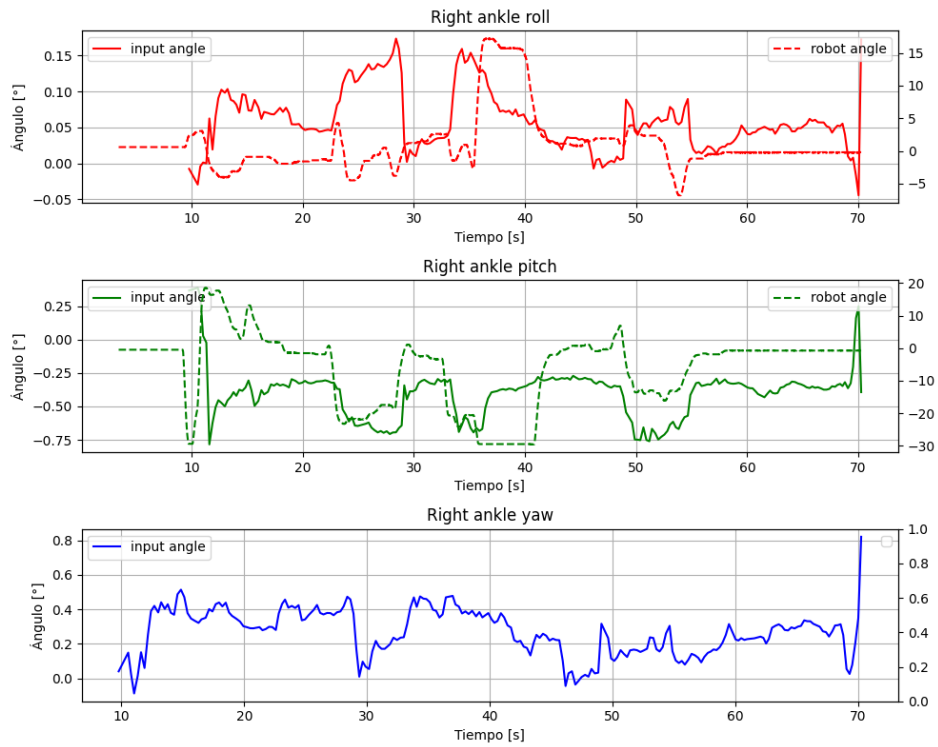


Figura 5.37: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el tobillo derecho.

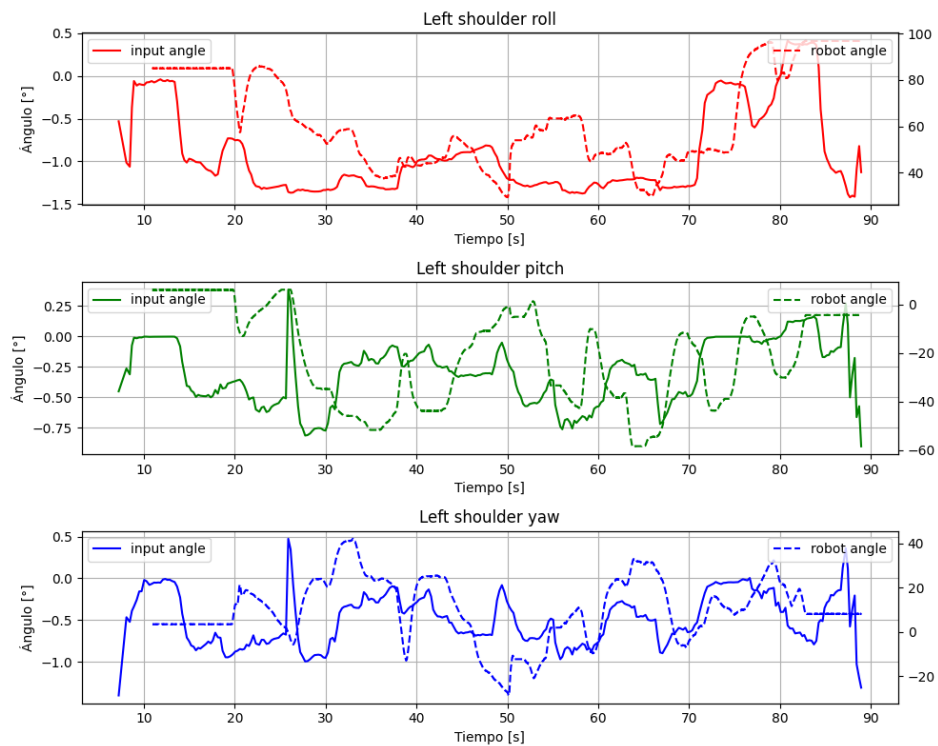


Figura 5.38: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el hombro izquierdo.

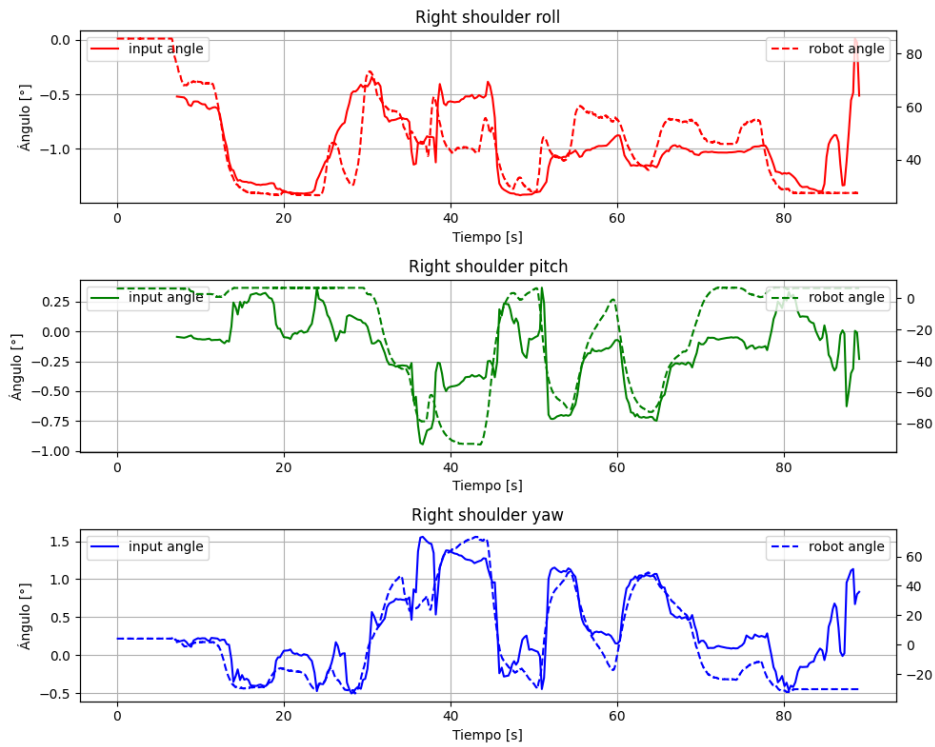


Figura 5.39: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el hombro derecho.

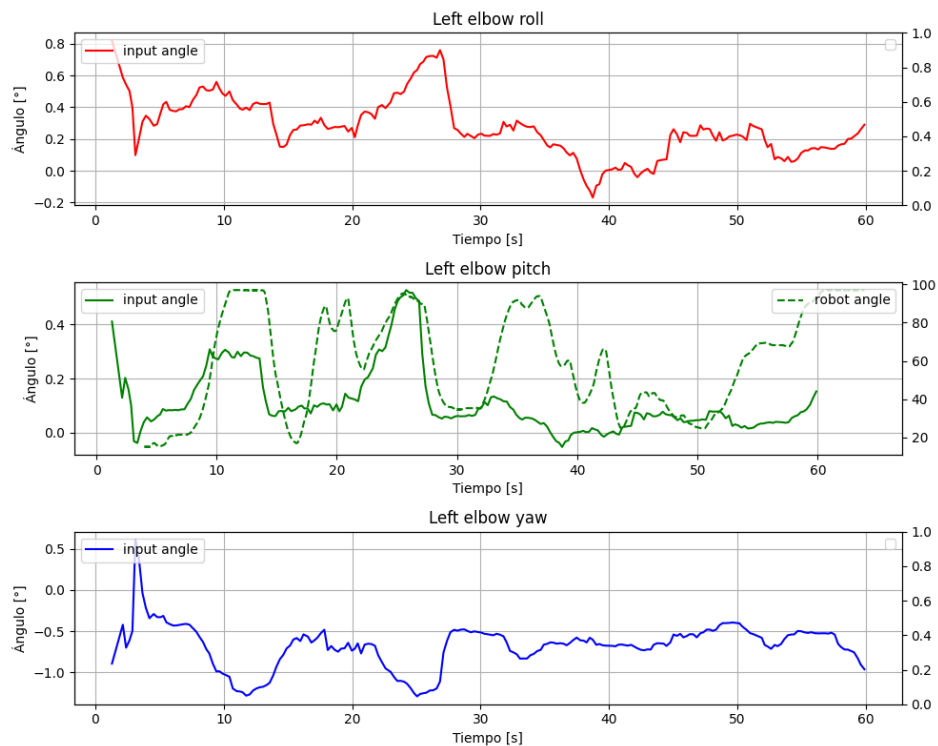


Figura 5.40: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el codo izquierdo.

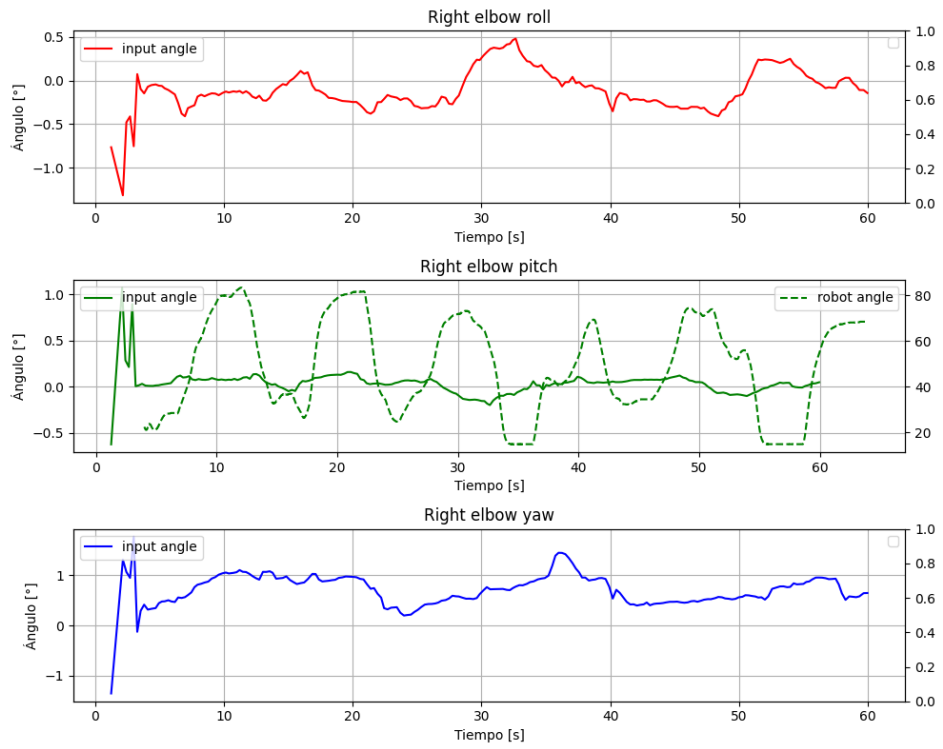


Figura 5.41: Comparación entre ángulos de entrada y ángulo alcanzado por el robot para el codo derecho.

Capítulo 6

Conclusiones

En este informe se presentó lo que será el trabajo de investigación “Teleoperación de robot iCub en tiempo real”. En esta investigación, se logra desarrollar un sistema de teleoperación en tiempo real basado en visión computacional para la teleoperación de un robot humanoide, en particular un robot iCub versión 2.7. En particular, se logra implementar un sistema de detección de poses humanas presentes en un video en vivo y que sea capaz de obtener información de posición y velocidad de sus principales articulaciones. También, se logra modificar un controlador existente para la pose del robot, de modo que este sea capaz de funcionar sin el uso de sensores de movimientos para la captura de información cinemática del operador. Para la comunicación de ambos módulos desarrollados, se logra implementar un sistema de comunicación indiferente a los lenguajes de programación utilizados en cada subsistema, logrando así obtener un sistema completo que permite la teleoperación de un robot utilizando como entrada el video captado por una cámara web en tiempo real. El sistema desarrollado se valida en un robot real, comprobando así su funcionamiento. A continuación se revisan los principales tópicos y hallazgos de cada capítulo de este trabajo.

En el *Capítulo 2: Marco teórico*, se estudian todos los conceptos y materias necesarias para el desarrollo del trabajo. Se define el robot iCub, indicando sus características físicas principales, estudiando los subsistemas que lo componen, su capacidad de movimiento y su forma de programación. Luego se estudia YARP, que corresponde al *software* con el que se realizará toda la programación y control del iCub. Para lograr la teleoperación utilizando como referencia información visual, se investiga qué son y cómo funcionan los algoritmos de detección de poses humanas, cómo se caracteriza una pose humana y qué soluciones existen actualmente a este problema. Finalmente, se repasan los conceptos de cinemática directa e inversa y estabilidad estática y dinámica, materias claves en el estudio de la robótica.

En el *Capítulo 3: Estado del arte*, se revisa la literatura disponible y las soluciones actuales a problemas similares al de investigación. En *Simultaneous Floating-Base Estimation of Human Kinematics and Joint Torques* (Sección 3.1) se presenta un método estocástico para la resolución del problema de cinemática inversa y la estimación simultánea de la cinemática y dinámica de un cuerpo humano con la pelvis como base flotante. En el artículo *Whole-Body Geometric Retargeting for Humanoid Robots* estudiado en la sección 3.2 se desarrolla un *framework* para la teleoperación de robots humanoides utilizando un método de reajuste de movimiento escalable, permitiendo que la teleoperación pueda ser hecha por diferentes sujetos humanos de diferentes dimensiones.

Finalmente, en el artículo *Model-Based Real-Time Motion Tracking Using Dynamical Inverse Kinematics* (Sección 3.3) se revisa un método de optimización en el cómputo de las estimaciones de la cinemática y dinámica de los modelos humanos y humanoides, obteniendo un mejor desempeño computacional para casos con modelos altamente articulados y críticos de tiempo. De esta revisión, el artículo *Simultaneous Floating-Base Estimation of Human Kinematics and Joint Torques* junto con el repositorio de *GitHub human-dynamics-estimation* conformarían las bases para la solución desarrollada.

En el *Capítulo 4: Diseño e implementación del sistema* se definen los requerimientos del sistema solución a desarrollar, sus elementos, alcances y limitaciones. Se propone un sistema compuesto por cuatro subsistemas: Visión, encargado de el reconocimiento y detección de poses; Transformación de datos, encargado de estimar los valores faltantes para poder simular la información generada por sensores de movimiento; Control, sub sistema encargado de recibir desde una nueva fuente (diferente a los sensores de movimiento) la información cinemática que le permita replicar una pose y controlar los movimientos del robot; y el sistema de Validación, encargado de obtener valores cuantitativos de los diferentes experimentos a realizar para su posterior evaluación.

En el *Capítulo 5: Evaluación y análisis de resultados*, se definen y registran las pruebas realizadas y los resultados obtenidos para cada experimento. En primera instancia se evalúa el funcionamiento de cada subsistema por separado, obteniendo resultados positivos y tiempos de inferencia del orden de milisegundos, logrando reconstruir poses a partir de los ángulos estimados de la pose detectada, logrando la integración y comunicación de sub sistemas escritos en diferentes lenguajes de programación y, logrando definir un controlador del robot iCub capaz de funcionar sin el uso de sensores de captura de movimiento.

Una vez validados los subsistemas, se evalúa el desempeño del sistema general tanto en simulación como utilizando el robot real. Para ambos casos, los resultados son bastante similares, lográndose la teleoperación del robot pero con diferencias entre la pose objetivo y la pose lograda. Estas diferencias se atribuyen a la diferencia física entre el cuerpo del humano y del robot, ya que el robot intentará alcanzar la pose más cercana a la realizada por el operador pero estando limitado por sus características de *hardware*, y también se puede atribuir a variaciones en la aproximación de los ángulos estimados en el subsistema de *Transformación de datos*. Finalmente, el funcionamiento del sistema completo se evalúa de forma cuantitativa utilizando los valores registrados por los *encoders* del robot.

El trabajo realizado presenta una metodología de teleoperación de robots humanoides basada en visión computacional. Se espera que este trabajo sea una contribución a la investigación y desarrollo de tecnologías para la teleoperación de maquinaria, eliminando la limitación del uso de sensores, ya sea de profundidad o de captura de movimiento, y que incentive la investigación en el campo de detección de poses humanas utilizando visión computacional.

Apéndice A

Robot iCub: especificaciones técnicas

A.1. Versiones.

La Tabla A.1 contiene las versiones disponibles del robot iCub y sus principales características. Esta corresponde a la traducción de la tabla de versiones disponible en la documentación oficial del robot iCub, disponible en el sitio web https://icub-tech-iit.github.io/documentation/icub_versions/#currently-known-icub-versions.

Tabla A.1: Versiones disponibles del iCub y sus principales características.

iCub - versión	Características
v1.0	Esta versión fue distribuida durante la convocatoria abierta de RobotCub. Posee las siguientes características: 1. Solo los brazos superiores (desde el hombro hasta el codo) están habilitados. 2. No hay sensores de fuerza disponibles. 3. No hay sensores táctiles disponibles.
v1.1	Las principales características son: 1. Articulaciones de la mano con sensores. 2. Sensores de fuerza de seis ejes en cada miembro. Esto permite el control de impedancia/compliance como se describe aquí. 3. Cuerpo totalmente habilitado (brazo superior, antebrazo, torso, cadera, piernas).
v1.1.1	Versión intermedia. La única diferencia con la v1.1 es que la placa de control principal ha cambiado de eCAN (o CFW1) a CFW2.
v1.2	Esta versión cuenta con sensores táctiles. Las principales características son: 1. Palma de la mano sensorizada. 2. Puntas de los dedos sensorizados.
v1.3	Esta versión cuenta con sensores táctiles añadidos en los antebrazos.
v1.4	Esta versión cuenta con sensores táctiles añadidos en los brazos superiores y pecho del robot.
v1.7	En esta versión se realizaron ligeras modificaciones en la cinemática de la muñeca.
v2.0	Esta versión tiene: 1. <i>Encoders</i> incrementales (ópticos) montados directamente en los motores (lado rápido del engranaje de reducción). 2. Conjuntos de antebrazo y manos completamente rediseñados. 3. Ensamblaje de cabeza completamente rediseñado para mejor rendimiento (motores del cuello más fuertes, movimiento ocular sin holgura). 3. Conjunto de piel sensible completamente montado.
v2.3	Corresponde a una versión v2.0 más cabeza parlante (ahora obsoleta, con motores para los labios y la mandíbula).
v2.5	Esta versión tiene: 1. Manos v2.0 2. Piernas más fuertes (con la posibilidad de montar series de elementos elásticos en la rodilla y en el tobillo)
v2.5.5	Corresponde a una versión v2.5 junto a una mochila con baterías.
v2.6	v2.5.5 con expresiones faciales nuevas (RFE master board)
v2.7	v2.6 con IMU de alto rendimiento en la cintura.
v2.8	v2.7 con divisor de haz en el ojo para alojar sensores orientados a eventos y basados en cuadros.
v2.9	v2.7 con manos mk5 y antebrazo mk2
v2.10	v2.7 con manos mk5, antebrazo mk2, nuevas cámaras basler 4k, NVIDIA Xavier e Intel i7 11 th en un módulo COM Express tipo 10 (UKIT009).

A.2. Rango de movimiento de articulaciones

En la Tabla A.2 se indican el límite inferior y superior de movimiento angular de cada articulación. Los límites se encuentran en grados.

Tabla A.2: Rango de movimiento de las articulaciones para distintas aplicaciones (*Hardware y Software*).

Articulación	Limites de Hardware	Límites de Software
Cuello - pitch	[-30, 22]	[-30, 22]
Cuello - roll	[-20, 20]	[-20, 20]
Cuello - yaw	[-45, 45]	[-45, 45]
Torso - yaw	[-50, 50]	[-50, 50]
Torso - roll	[-30, 30]	[-30, 30]
Torso - pitch	[-20, 70]	[-20, 70]
Hombro - pitch	[-95.5, 8]	[-95.5, 8]
Hombro - roll	[0, 160]	[0, 160]
Hombro - yaw	[-32, 80]	[-32, 80]
Codo	[15, 106]	[15, 106]
Cadera - pitch	[-30, 92]	[-30, 85]
Cadera - roll	[-15, 92]	[0, 85]
Cadera - yaw	[-72, 72]	[-70, 70]
Rodilla	[-100, 0]	[-100, 0]
Talón - pitch	[-30, 30]	[-30, 30]
Talón - roll	[-20, 20]	[-20, 20]

Bibliografía

- [1] Google, “MediaPipe Pose: Pose Estimation Quality..” <https://github.com/google/mediapipe/blob/master/docs/solutions/pose.md>, 2023.
- [2] K. Darvish, Y. Tirupachuri, G. Romualdi, L. Rapetti, D. Ferigo, F. J. A. Chavez, and D. Pucci, “Whole-body geometric retargeting for humanoid robots,” in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pp. 679–686, IEEE, 2019.
- [3] L. Rapetti, Y. Tirupachuri, K. Darvish, S. Dafarra, G. Nava, C. Latella, and D. Pucci, “Model-based real-time motion tracking using dynamical inverse kinematics,” *Algorithms*, vol. 13, no. 10, p. 266, 2020.
- [4] S. Gollapudi and S. Gollapudi, “Artificial intelligence and computer vision,” 2019.
- [5] IIT., “iCub history..” <https://icub.iit.it/about-us/icub-history>. [Accedido el 20 de diciembre, 2023].
- [6] “YARP: Welcome to YARP..” <https://www.yarp.it/latest/>. [Accedido el 20 de diciembre, 2023].
- [7] “iCub Software Repository Documentation..” <https://robotology.github.io/robotology-documentation/doc/html/index.html>. [Accedido el 20 de diciembre, 2023].
- [8] M. Vukobratović and B. Borovac, “Zero-moment point—thirty five years of its life,” *International journal of humanoid robotics*, vol. 1, no. 01, pp. 157–173, 2004.
- [9] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, “The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation,” vol. 1, pp. 239–246, 2001.
- [10] C. Latella, S. Traversaro, D. Ferigo, Y. Tirupachuri, L. Rapetti, F. J. Andrade Chavez, F. Nori, and D. Pucci, “Simultaneous floating-base estimation of human kinematics and joint torques,” *Sensors*, vol. 19, no. 12, p. 2794, 2019.
- [11] Google, “MediaPipe - Open source cross-platform ML solutions.” <https://github.com/google/mediapipe>, 2023.
- [12] Temuge Batpurev, “Estimating joint angles from 3D body poses,” 2021. [Accedido el 20 de

diciembre, 2023].

- [13] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, “Blazepose: On-device real-time body pose tracking,” *arXiv preprint arXiv:2006.10204*, 2020.
- [14] Google, “Pose landmark detection guide for web.” https://developers.google.com/mediapipe/solutions/vision/pose_landmarker/web_js, 2024.