UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Control del robot humanoide NAO mediante poses humanas captadas por cámara

Trabajo de graduación presentado por Sergio Alejandro De León Orellana para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

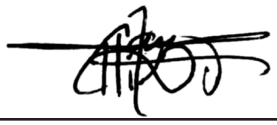
Control del robot humanoide NAO mediante poses humanas captadas por cámara

Trabajo de graduación presentado por Sergio Alejandro De León Orellana para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2022

Vo.Bo.:



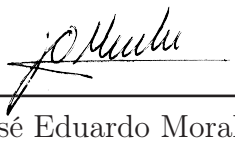
(f) _____

Ing. Carlos Esquit

Tribunal Examinador:

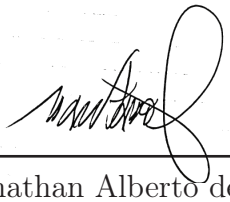
(f) _____

Ing. Carlos Esquit



(f) _____

Ing. José Eduardo Morales Espinoza



(f) _____

Ing. Jonathan Alberto de los Santos Chonay

Fecha de aprobación: Guatemala, 7 de enero de 2022.

Índice

Lista de figuras

Lista de cuadros

Resumen

La interacción humano-robot es un campo de investigación bastante importante, ya que promueve la autonomía de los robots. En este trabajo se propone un sistema que permite a al robot humanoide NAO, imitar movimientos del cuerpo humano en tiempo real. Esto para contar con una manera interactiva de controlar el robot.

Actualmente la Universidad del Valle de Guatemala cuenta con acceso a dos robots NAO, estos han sido de bastante utilidad para despertar el interés de estudiantes hacia el estudio de la robótica y la carrera de ingeniería en sí. Debido a gran nivel de interactividad lo han convertido en un elemento ideal para proyectos de publicidad de la Universidad y de la carrera de ingeniería mecatrónica. Por lo que desarrollar rutinas para tener un mejor control del robot NAO es de bastante interés. Para esto se propone una herramienta capaz de interactuar con el robot por medio de una cámara, logrando que el robot imite las poses realizadas por una persona. Esto facilitaría la programación del robot para trayectorias y rutinas más complejas reemplazando la forma tradicional de imitación de movimiento con el método de paso directo, en donde el programador coloca el robot como se desee y lo instruye con una secuencia de poses que constituye el movimiento del robot.

Se tomará ventaja de los avances sobre reconocimiento de imágenes, ya que cada vez existen algoritmos más potentes en base a machine learning capaces de reconocer diversos aspectos de las expresiones del cuerpo humano. Esto es de bastante utilidad al momento de reconocer y procesar las distintas poses que puede realizar el cuerpo humano. De esta manera se puede realizar el reconocimiento de imágenes por medio de una cámara promedio que facilitara la versatilidad para utilizar la herramienta. Ya que cualquier cámara se puede utilizar como dispositivo para capturar los movimientos humanos en tiempo real, solo se necesita un algoritmo especializado para hacer que el robot humanoide imite el movimiento actual.

CAPÍTULO 1

Introducción

En la actualidad se llevan a cabo diversos esfuerzos en proyectos de investigación en cuanto a robots humanoides se refiere, debido a que analizar el funcionamientos de estos robots contribuye a la innovación de nuevas tecnologías capaces de ser adaptables al cuerpo humano como por ejemplo las prótesis. Varios estudios de robótica se han realizado con base en la plataforma NAO, este es un robot humanoide de 58 centímetros, interactivo, totalmente programable y de una gran complejidad. Este cuenta con distintos niveles para su programación para que este se adapte a usuarios de cualquier nivel, ya sea principiantes por medio de su software Choreographe o también para niveles superiores o investigadores con Python o C++.

Se han realizado distintos proyectos en los cuales se logra la imitación del robot en tiempo real, sin embargo se necesita de equipo especializado como dispositivos destinados a la captura y percepción del movimiento humano en tiempo real. Por lo que se trabajó con inteligencia artificial para reconocer las poses humanas y solamente necesitar una cámara común para captar el movimiento. Para obtener datos del movimiento humano solo es cuestión de desarrollar un algoritmo que suavice los datos y los traduzca para que sean reproducidas por las juntas del robot.

CAPÍTULO 2

Justificación

El robot humanoide NAO es una pieza de mucha utilidad para el aprendizaje e interés temprano de los estudiantes que entran a una carrera de ingeniera, el poder hacer manejo interactivo del robot a través de los propios movimientos de una persona, será una herramienta de utilidad al momento de querer realizar proyectos con el robot. Esto se puede utilizar para proyectos de publicidad para la facultad, ya que se tiene un control más directo e interactivo sobre el robot.

Además el uso de inteligencia artificial para el reconocimiento de las poses humanas en las imágenes, hará que la herramienta sea más versátil ya que no se necesita de equipo especializado para captar las poses de la persona y lograr la imitación en tiempo real del robot.

CAPÍTULO 3

Objetivos

3.1. Objetivo general

Diseñar un algoritmo para que el robot humanoide NAO logre imitar las poses humanas captadas mediante una cámara usando inteligencia artificial.

3.2. Objetivos específicos

- Lograr el seguimiento de poses del cuerpo humano mediante puntos de referencia de alta fidelidad cuerpo completo.
- Usar la plataforma Choregraphe 2.1.4 para la programación de NAO.
- Realizar el procesamiento de video e imágenes mediante inteligencia artificial para obtener las poses del cuerpo.
- Captar las poses del cuerpo mediante una cámara.
- Traducir las poses obtenidas en ángulos que pueda usar las juntas del robot humanoide NAO.
- Crear restricciones en la imitación de las poses para evitar exceder el rango de movimiento del robot NAO.
- Hallar la solución de dinámica directa del efector final del robot NAO.
- Reconstruir las trayectorias de movimiento con alta credibilidad.
- Lograr la imitación en tiempo real satisfactoria y segura en un robot humanoide
- Refinar los datos de captura de movimiento suavizando las trayectorias de movimiento.

CAPÍTULO 4

Marco teórico

4.1. Imitación del movimiento humano en robots humanoides

Los robots humanoides han atraído mucho la atención en la ultima década debido a la amplia aplicación que tienen para ser utilizados en el área de investigaciones científicas. Entre las áreas de interés esta la imitación del movimiento humano, un tema bastante popular ya que se utiliza como un principio para estudiar el tema mecánico. Se han realizado varios estudios sobre la imitación del movimiento de un robot. Para esto se han desarrollado diversas herramientas para captar el movimiento humano tales como giroscopios, cámaras infrarrojas como el Kinect y Xsens. Recientemente incluso se han desarrollado técnicas de inteligencia artificial capaz de reconocer el movimiento humano utilizando solamente imágenes [1].

4.2. Cinemática de robots

La cinemática de robots es la aplicación de la geometría al estudio de cadenas cinemáticas con múltiples grados de libertad. Más específicamente, la cinemática del robot proporciona la transformación del espacio articular, donde se las cadenas cinemáticas se definen, al espacio cartesiano, donde se mueve el manipulador del robot, y viceversa. La cinemática del robot es bastante útil, ya que se puede utilizar para planificar y ejecutar movimientos, así como para calcular fuerzas y momentos de los actuadores [2].

Una cadena cinemática de robot es un manipulador articulado que interactúa con el entorno y se describe típicamente como un conjunto de enlaces conectados por juntas (giratorias). Las articulaciones giran y controlan la posición angular relativa de los enlaces del manipulador. No todas las combinaciones de posiciones de uniones en la cadena son válidas, porque algunas combinaciones dan lugar a colisiones entre los eslabones de la cadena o con algún elemento fijo del entorno, como el suelo o una pared. Todas las combinaciones válidas de valores articulares forman el espacio articular.El término "grados de libertad"(GDL) se refiere al número de articulaciones en una cadena cinemática; a más GDL implican más

flexibilidad en el movimiento de la cadena[2].

4.2.1. Cinemática directa

La cinemática directa se refiere a los datos cinemáticos para las coordenadas de la articulación que se utilizan para encontrar los datos en el marco de coordenadas cartesianas base [3]. El análisis de la cinemática directa es un problema relativamente sencillo. Claramente, existe una pose única para un conjunto específico configuraciones de las articulaciones del robot [4].

La cinemática directa obtiene el posicionamiento y orientación del efector final. El posicionamiento es llevar el efector final a un punto arbitrario dentro de la dextrosa, mientras que la orientación es mover el efector final a la orientación requerida en la posición. Para simplificar el análisis cinemático, podemos desacoplar el posicionamiento y la orientación del efector final. En términos de la formación cinemática, un robot de 6 GDL comprende seis enlaces móviles secuenciales y seis articulaciones con al menos los dos últimos enlaces de longitud cero. En términos generales, casi todos los problemas de la cinemática se pueden interpretar como una suma de vectores. Sin embargo, todos los vectores de una ecuación vectorial deben transformarse y expresarse en un marco de referencia común [3].

4.2.2. Cinemática inversa

La cinemática inversa se refiere a los datos cinemáticos para el efector final en el espacio cartesiano que se utilizan para encontrar los datos cinemáticos en el espacio articular.La cinemática inversa es muy no lineal y, por lo general, es un problema mucho más difícil que el problema de la cinemática directa. Los problemas de velocidad inversa y aceleración son lineales y mucho más simples una vez que se ha resuelto el problema de la posición inversa [3].

4.3. Transformación afín

Una transformación afín es un mapeo que transforma puntos y vectores de un espacio a otro, de una manera que preserva las proporciones de distancias. Los espacios de origen y destino pueden ser n-dimensionales donde n tiene que ser mayor a 2. Las siguientes son transformaciones afines: contracción geométrica, expansión, dilatación, reflexión, rotación, transformaciones de similitud, similitudes espirales y traslación. Todas las combinaciones posibles de lo anterior producen una transformación afín también. La flexibilidad de una nueva transformación con respecto a la representación de objetos en diferentes espacios, la convierte en una herramienta muy útil en información gráfica procesada por computadora [2].

En este trabajo vamos a enfocarnos en dos tipos de transformaciones, rotación y traslación, por lo que nos enfocaremos solo en estos dos tipos de transformación afín. Adicionalmente, estamos trabajando en un espacio de trabajo cartesiano tridimensional y por lo

tanto todas las definiciones y ejemplos a partir de ahora se centrarán en este espacio [4].

4.3.1. Matrices de transformación afín

Una matriz de transformación afín es una matriz (n + 1)x(n + 1), donde n es el número de dimensiones en el espacio en el que se define la transformación. En general, una matriz de transformación afín es una matriz de bloques de la forma 1:

$$T = \begin{bmatrix} X & Y \\ [0 \ ... \ 0] & 1 \end{bmatrix} \tag{1}$$

donde X es una matriz (nxn), Y es un vector (nx1) y la última línea de T contiene n-1 ceros seguidos de un 1. Si queremos aplicar la transformación, a un punto dado $p = (p_1, p_2, ..., p_n)$ en el espacio n-dimensional, simplemente multiplicamos la matriz de transformación afín con el vector columna $v = (p_1, p_2, ..., p_n)^T$ 2:

$$v' = \begin{bmatrix} p'_1 \\ . \\ . \\ . \\ p'_n \\ 1 \end{bmatrix} = T_v \begin{bmatrix} X & Y \\ [0 \ ... \ 0] & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ . \\ . \\ . \\ p_n \\ 1 \end{bmatrix} \tag{2}$$

Para un punto $p = (p_x, p_y, p_z)$ en el espacio tridimensional, la transformación será 3:

$$v' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = T_v \begin{bmatrix} X_{xx} & X_{xy} & X_{xz} & Y_x \\ X_{yx} & X_{yy} & X_{yz} & Y_y \\ X_{zx} & X_{zy} & X_{zz} & Y_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{3}$$

La matriz que resulta de la multiplicación de dos matrices de transformación afín T_1 y T_2 sigue siendo una transformación afín 4:

$$T = T_1 T_2 = \begin{bmatrix} X_1 & Y_1 \\ [0 \ ... \ 0] & 1 \end{bmatrix} \begin{bmatrix} X_2 & Y_2 \\ [0 \ ... \ 0] & 1 \end{bmatrix} = \begin{bmatrix} X_1 X_2 & X_1 X_2 + Y_1 \\ [0 \ ... \ 0] & 1 \end{bmatrix} \tag{4}$$

Esta propiedad se generaliza al producto de cualquier número de matrices de transformación afín:

$$\hat{T} = T_1 T_2 T_3 ... T_i = \begin{bmatrix} \hat{X} & \hat{Y} \\ [0 \ ... \ 0] & 1 \end{bmatrix} \tag{5}$$

[2]

Una matriz de transformación afín es invertible, si y solo si X es invertible, y toma la forma:

$$T^{-1} = \begin{bmatrix} X^{-1} & -X^{-1}Y \\ [0 \ ... \ 0] & 1 \end{bmatrix} \tag{6}$$

4.3.2. Matrices de traslación

La traslación en un espacio cartesiano es una función que mueve (traslada) cada punto una distancia fija en una dirección especificada. Podemos describir una traslación en el espacio tridimensional con una matriz (4x4) de la siguiente forma 7:

$$A = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

donde d_x, d_y, d_z definen la distancia de traslación a lo largo de los ejes x, y y z, respectivamente. Aparentemente, la matriz de traslación es una matriz de transformación afín con X = I. Por lo tanto, para mover un punto $p = (p_x, p_y, p_z)$ en el espacio tridimensional por distancias (d_x, d_y, d_z), simplemente aplicamos la transformación 8:

$$v' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = A_v = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{8}$$

4.3.3. Matrices de rotación

La rotación en un espacio cartesiano es una función que rota los vectores en un ángulo fijo alrededor de una dirección especificada. Una rotación en el espacio n-dimensional se describe como una matriz ortogonal (nxn) R con determinante 1:

$$R^T = R^{-1} \tag{9}$$

$$RR^T = R^T R = 1 \tag{10}$$

$$det(R) = 1 \tag{11}$$

En el espacio cartesiano tridimensional hay tres matrices de rotación distintas, cada una de las cuales realiza una rotación de $\theta_x, \theta_y, \theta_z$ alrededor del eje x, y, z respectivamente, asumiendo un sistema de coordenadas conforme a las agujas del reloj. [2]

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix} \tag{12}$$

$$R_y = \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & x & cos(\theta_y) \end{bmatrix} \tag{13}$$

$$R_z = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

Para rotar un vector definido por el punto final $p = (p_x, p_y, p_z)$ alrededor de un eje específico, uno puede simplemente multiplicar con la matriz de rotación correspondiente. Para rotar el vector primero sobre el eje xy luego sobre el eje y, hay que multiplicar con las matrices de rotación correspondientes en el siguiente orden:

$$p' = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = R_y R_x p = \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & x & cos(\theta_y) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{15}$$

Por lo que se puede notar que las tres matrices de rotación se pueden combinar para formar nuevas matrices de rotación para realizar rotaciones complejas en todas las dimensiones.Por ejemplo, la matriz de rotación que rota los vectores primero sobre el eje x, luego sobre el eje y y finalmente sobre el eje z es la siguiente:

$$R' = R_z R_y R_z \tag{16}$$

La forma analítica de la matriz de rotación anterior es la siguiente:

$$R' = \begin{bmatrix} cos(\theta_y)cos(\theta_z) & -cos(\theta_x)sin(\theta_z) + sin(\theta_x)sin(\theta_y)cos(\theta_z) & sin(\theta_x)sin(\theta_z) + cos(\theta_x)sin(\theta_y)cos(\theta_z) \\ cos(\theta_y)sin(\theta_z) & cos(\theta_x)cos(\theta_z) + sin(\theta_x)sin(\theta_y)sin(\theta_z) & -sin(\theta_x)cos(\theta_z) + cos(\theta_x)sin(\theta_y)sin(\theta_z) \\ -sin(\theta_y) & -sin(\theta_x)cos(\theta_y) & cos(\theta_x)cos(\theta_y) \end{bmatrix} \tag{17}$$

Podemos transformar fácilmente cualquier matriz de rotación $\hat{R}$ en una matriz de transformación afín R simplemente rellenando la última línea y la última columna con (0,..., 0, 1) 18.Por lo que se puede decir que cualquier matriz de rotación será una nueva matriz de transformación.

$$R = \begin{bmatrix} & & & [0 \\ & \hat{R} & & . \\ & & & . \\ & & & . \\ & & & 0] \\ [0 & . \ . & . \ . & 0] & 1 \end{bmatrix} \tag{18}$$

A los efectos de la cinemática, estamos usando matrices de rotación y traslación, de modo que podamos transformar puntos en el espacio tridimensional. Consideramos una nueva transformación de matrices que combinan rotación y traslación; el bloque X de la matriz define la rotación, mientras que el bloque Y de la matriz define la traslación [19]:

$$T = \begin{bmatrix} & \hat{R} & & [d_x] \\ & & & [d_y] \\ & & & [d_z] \\ [0 & 0 & 0] & 1 \end{bmatrix} \tag{19}$$

[2]

4.4. Parámetros de Denavit-Hartenberg (DH)

El método Denavit-Hartenberg (DH) de asignar marcos de coordenadas relativas de los enlaces de un robot, es el método más común utilizado. Este se usa para la definición de mecanismos espaciales y en la transformación homogénea de puntos. La matriz 4×4 o la transformación homogénea se utiliza para representar transformaciones espaciales de vectores puntuales. En robótica, esta matriz se utiliza para describir un sistema de coordenadas con respecto a otro. El método de la matriz de transformación es la técnica más popular para describir los movimientos del robot. [3].

Este método consiste en crear una matriz de transformación que describe puntos en un extremo de una articulación a un sistema de coordenadas que se fija al otro extremo de la articulación, en función del estado de la articulación. De manera que podemos describir completamente esta matriz de transformación utilizando solo cuatro parámetros, conocidos como parámetros de Denavit-Hartenberg (DH): a,α, d,θ. Antes de explicar estos parámetros, primero se debe establecer el marco de referencia de cada articulación i con respecto al marco de referencia de su articulación anterior:[2]

- El eje z_i se establece en la dirección del eje de la articulación (la dirección de rotación).
- El eje x_i es paralelo a la normal común entre z_i y z_{i-1} (producto exterior). La dirección de x_i se deriva usando la regla de la mano derecha de z_{i-1} a z_i.

Ahora, podemos describir los parámetros DH ilustrados en la Figura 1:

[3]

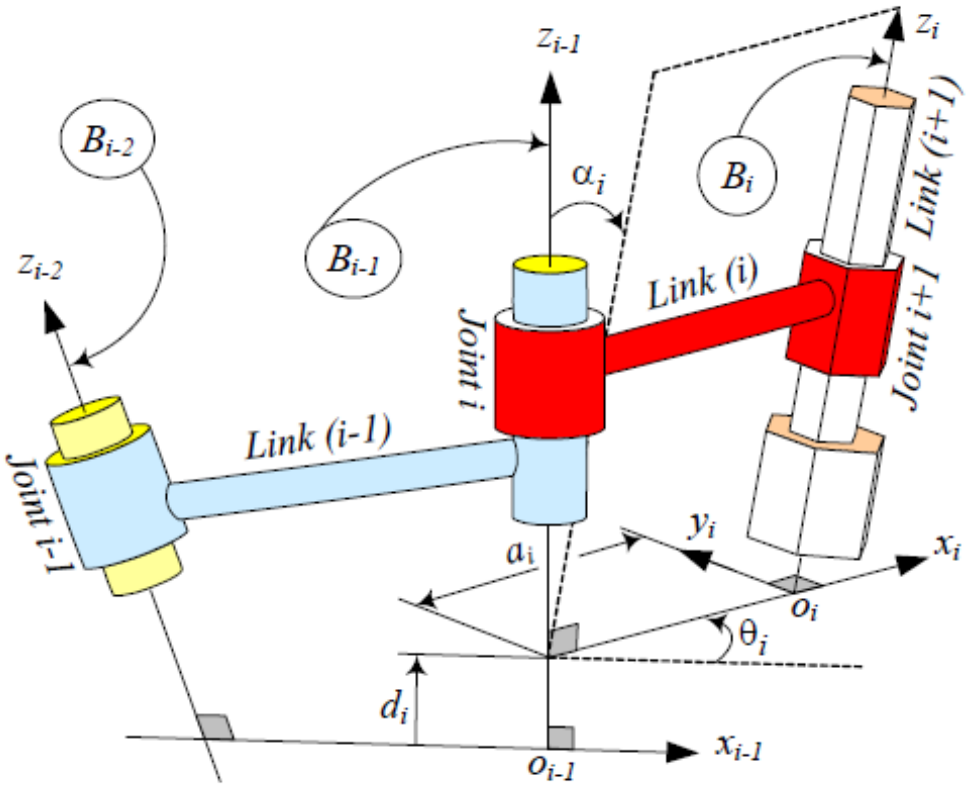


Figura 1: Parámetros DH $a_1, \alpha_i, d_i, \theta_i$ definidos por las juntas i y el enlace (i).

- a_i: longitud de la normal común.
- α_i: ángulo sobre la normal común, desde el eje z_{i-1} al eje z_i.
- d_i: Desplazamiento a lo largo del eje z_{i-1} la normal común.
- θ_1: Ángulo sobre el eje z_{i-1}desde el eje x_{i-1}al eje x_i.

[2]

4.5. NAO SoftBank Robotics

NAO es un robot humanoide integrado, programable y de tamaño mediano desarrollado por Aldebaran Robotics en París, Francia. El proyecto NAO comenzó en 2004. Es el primer robot creado por SoftBank Robotics. Es bastante popular debido a que es una excelente herramienta de programación que se ha convertido en todo un referente en los ámbitos de la educación y la investigación [5]. La versión de NAO que posee la Universidad del Valle de Guatemala es la V5 [2].

NAO está compuesto de 21 grados de libertad; 2 en la cabeza, 4 en cada brazo, 5 en cada pierna y 1 en la pelvis (hay dos articulaciones de la pelvis que están acopladas en un servo y no pueden moverse de forma independiente). NAO, también, cuenta con una variedad de sensores. Dos cámaras están montadas en la cabeza en alineación vertical que brindan vistas no superpuestas de las áreas frontales inferiores y distantes, pero solo una está activa cada vez y la vista se puede cambiar de una a otra casi instantáneamente. Cada cámara es un dispositivo VGA 640 480 que funciona a 30 fps. Cuatro sonares (dos emisores y dos receptores) en el cofre permiten que NAO detecte obstáculos frente a él. Además, el NAO tiene una rica unidad inercial, con un giroscopio de 2 ejes y un acelerómetro de 3 ejes, en el torso que proporciona información en tiempo real sobre sus movimientos corporales instantáneos. Dos parachoques ubicados en la punta de cada pie son simples interruptores de ENCENDIDO / APAGADO y pueden proporcionar información sobre colisiones de los pies con obstáculos. Finalmente, una serie de resistencias sensibles a la fuerza en cada pie proporciona retroalimentación de las fuerzas aplicadas a los pies, mientras que los codificadores en todos los servos registran los valores reales de todas las articulaciones en cada momento [5].

[6]

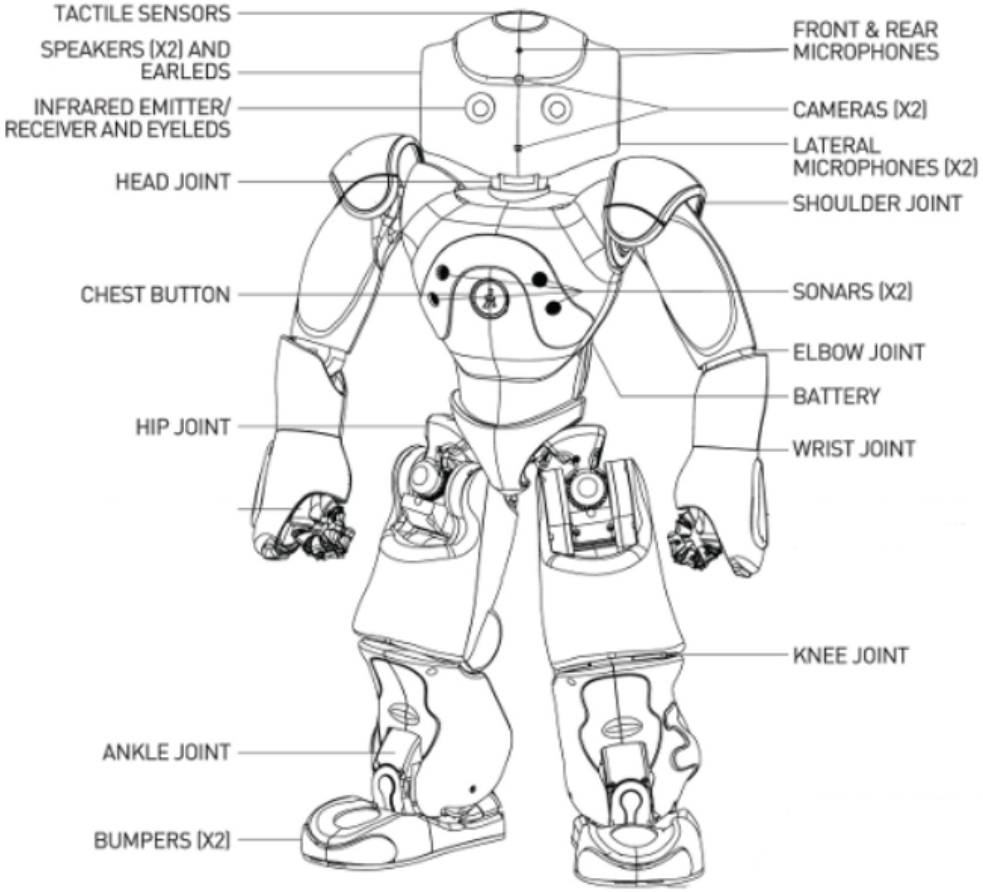


Figura 2: Componentes de Aldebaran NAO v5

CAPÍTULO 5

Análisis

5.1. Especificaciones del robot NAO

Nao Albedaran es un robot humanoide con cinco cadenas cinemáticas (cabeza, dos brazos, dos piernas). Tiene una altura de 58cm y aproximadamente con una masa de 5kg. La versión con la que se esta trabajando es la v5 con 21 grados de libertad(GDL). Nao tiene dos GDL en la cabeza, cuatro GDL en cada brazo, cinco GDL en cada pierna y un GDL en la pelvis que esta entre las piernas. Las cinco cadenas cinemáticas y sus articulaciones son los siguientes:

Head: HeadYaw, HeadPitch

Left Arm: LShoulderPitch, LShoulderRoll, LElbowYaw, LElbowRoll

Right Arm: RShoulderPitch, RShoulderRoll, RElbowYaw, RElbowRoll

Left Leg: LHipYawPitch, LHipRoll, LHipPitch, LKneePitch, LAnklePitch, LAnkleRoll

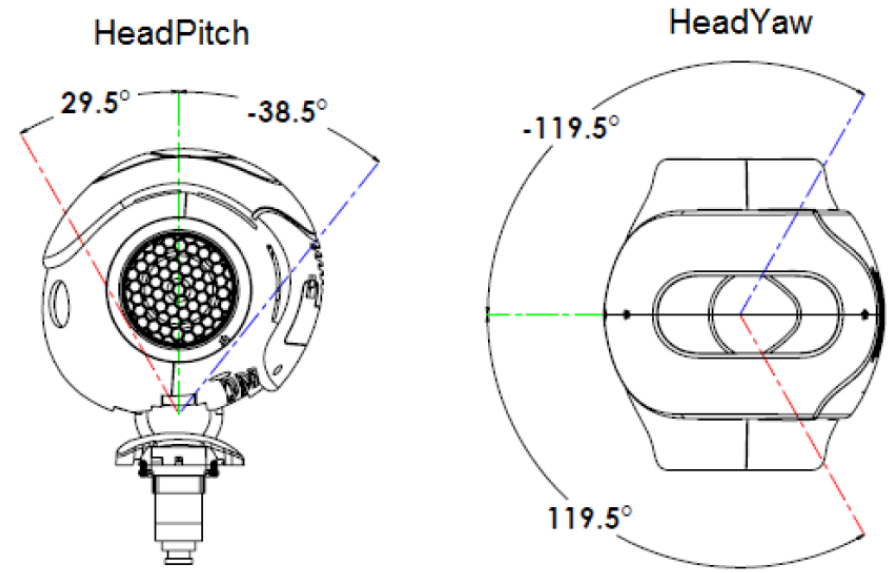
Right Leg:RHipYawPitch, RHipRoll, RHipPitch, RKneePitch, RAnklePitch, RAnkleRoll

Se utilizaran las articulaciones de los brazos y la cabeza para imitar el movimiento. El listado de las juntas de la cabeza y sus rangos se muestran en el Cuadro 1 y se ilustran en 3.De la misma manera el listado de juntas de los brazos y sus rangos se muestran en el Cuadro 2 y se ilustran en 4.

Cuadro 1: Rango de movimiento cabeza NAO

[6]

Nombre de articulación	rango en grados	rango en radianes
HeadYaw	-119.5° a 119.5°	-2.0857 a 2.0857
HeadPitch	-38.5° a 29.5°	-0.6720 a 0.5149



[6]

Figura 3: Articulaciones y rango de movimiento de la cabeza de NAO

5.1.1. Complejidad de la cinemática para NAO

Nao es robot con un gran numero de GDL, por eso mismo puede ejecutar una gran variedad de movimientos complejos como caminar, ponerse de pie, manipular objetos, etc. El estudio del problema cinemático del robot es muy útil ya que nos permite crear soluciones al momento de planear y ejecutar movimientos deseados.

Problema de cinemática directa para NAO

El problema de cinemática directa consiste en definir un mapeo desde el espacio de las juntas del robot al espacio tridimensional con respecto a cualquier marco de coordenadas base. A pesar de los 21 grados de libertad que maneja NAO, el problema de cinemática directa se puede descomponer fácil, ya que tres de las cinco cadenas (cabeza y ambos brazos) son totalmente independientes de las otras dos (ambas piernas). Luego, tomando encuentra que la cinemática directa no afecta los valores de las juntas del robot, sino que solamente necesita leer sus valores, podemos considerar que incluso ambas piernas son independientes (ignorando el ángulo de la pelvis). Por esto mismo la cinemática directa de NAO se puede dividir en cinco problemas independientes con sus respectivas soluciones por cada cadena. Cada solución proporciona la posición y orientación en el espacio tridimensional del efector

Cuadro 2: Rango de movimiento brazos NAO

[6]

Nombre de articulación	Rango en Grados	Rango en Radianes
LShoulderPitch	-119.5° a 119.5°	-2.0857 a 2.0857
LShoulderRoll	-18° a 76°	-0.3142 a 1.3265
LElbowYaw	-119.5° a 119.5°	-2.0857 a 2.0857
LElbowRoll	-88.5° a -2°	-1.5446 a -0.0349
LWristYaw	-104.5° a 104.5°	-1.8238 a 1.8238
RShoulderPitch	-119.5° a 119.5°	-1.3265 a 0.3142
RShoulderRoll	-76° a 18°	-2.0857 a 2.0857
RElbowYaw	-119.5° a 119.5°	-2.0857 a 2.0857
RElbowRoll	2° a 88.5°	0.0349 a 1.5446
RWristYaw	-104.5° a 104.5°	-1.8238 a 1.8238
LWristYaw, RWristYaw	abierto y cerrado	abierto y cerrado

[6]

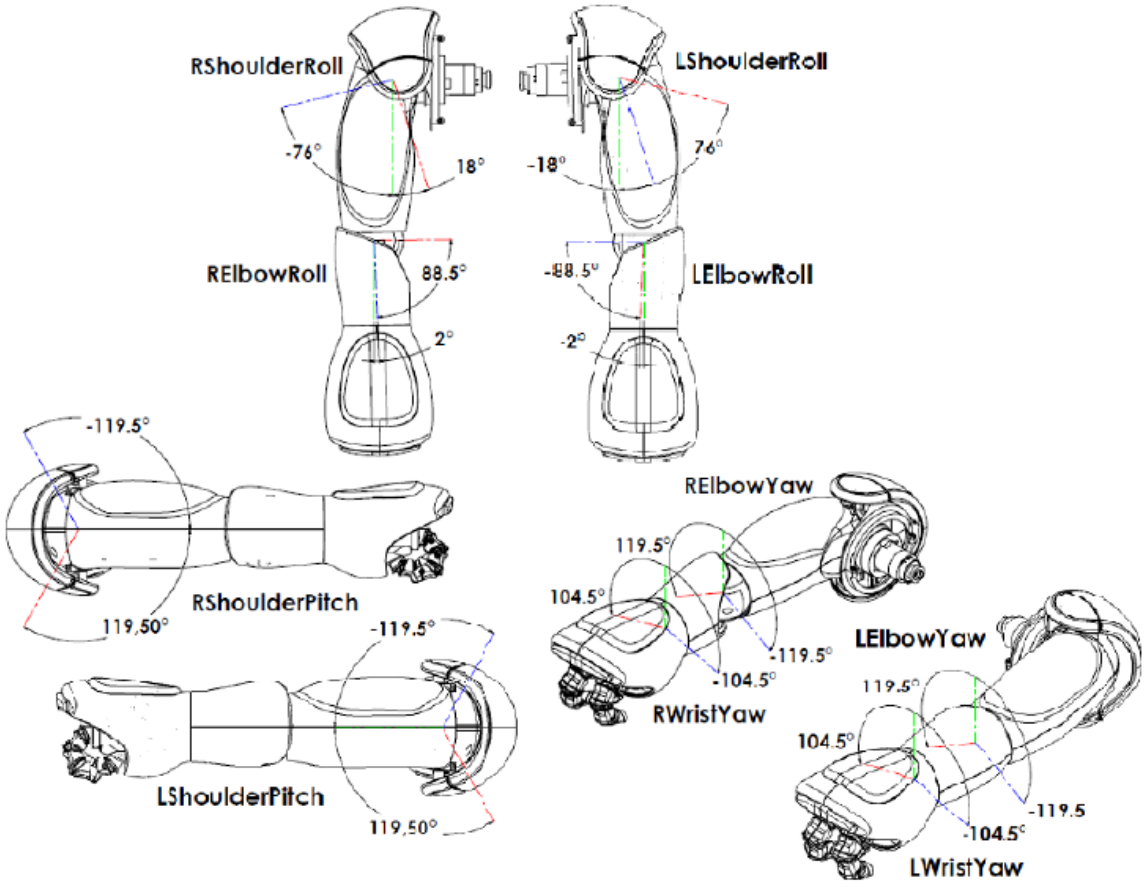


Figura 4: Articulaciones y rango de movimiento de los brazos de NAO

final de cada cadena. Estas soluciones se pueden combinar para obtener una solución para una cadena cinemática más grande formada por cualquier combinación de las cinco cadenas

independientes (por ejemplo, la cadena cinemática del pie derecho a la cabeza o la cadena cinemática del pie izquierdo a la mano derecha).

Otra importancia de resolver el problema de cinemática directa es que, además de la capacidad de localizar la posición y orientación exacta de cualquier efector final, esta proporciona los medio para calcular el centro de masa del robot para la configuración actual, lo cual es necesario a la hora de mantener el equilibro.

Problema de cinemática inversa para NAO

El problema de cinemática inversa consiste en definir un mapeo desde el espacio tridimensional a pasar al espacio articular del robot. En particular, define una relación entre puntos en el espacio tridimensional (posición y orientación) y valores articulares en el espacio articular de una cadena cinemática. Por lo expuesto el problema de cinemática inversa al igual que el problema de cinemática directa se puede descomponer en cinco problemas independientes. Se ignora inicialmente el acoplamiento que hay en la pelvis, ya que ambas articulaciones de las piernas lo comparten.

La cinemática inversa es un problema mas complejo en comparación con la cinemática directa, esto debido a que puede conducir a un sistema de ecuaciones no lineales que pueden o no tener una solución analítica, además también a medida que aumentan los GDLs y la cadena cinemática se vuelve mas flexible, un punto en el espacio tridimensional puede tener mas de una configuración en el espacio de la cadena. La multiplicidad de soluciones define una solución compleja pero no un mapeo entre los dos espacios.La importancia de resolver el problema de cinemática inversa para NAO radica en la capacidad de seguir cualquier trayectoria (predeterminada o generada dinámicamente) en el espacio tridimensional con cualquiera de los cinco e efectores finales. La cinemática inversa proporciona esencialmente el mecanismo para transformar dicha trayectoria en otra trayectoria en el espacio articular del robot. El problema de la cinemática inversa se puede resolver analíticamente con ecuaciones de forma cerrada o numéricamente con un método de aproximación iterativo. La solución analítica es en general más rápida que la solución numérica más rápida y, por lo tanto, es más apropiada para la ejecución en tiempo real. Las soluciones numéricas también están sujetas a singularidades, lo que hace que no se obtenga una solución, incluso si existe. Además, las soluciones numéricas son iterativas; para la ejecución en tiempo real, el número de iteraciones es limitado y, por lo tanto, es posible que no converjan. Por estas razones, nuestro objetivo es encontrar una solución analítica al problema de la cinemática inversa para el robot NAO. Es bien sabido que la cinemática inversa se puede obtener analíticamente si la cadena tiene cinco grados de libertad o menos. Si la cadena tiene seis GDL, se puede obtener una solución analítica, solo si tres uniones consecutivas tienen ejes que se cruzan. Tres de las cadenas cinemáticas del robot NAO tienen menos de cinco GDL. Las piernas tienen seis grados de libertad, sin embargo, cumplen con la condición dada anteriormente porque las tres articulaciones de la cadera tienen ejes que se cruzan. Por tanto, es posible obtener una solución completamente analítica para la cinemática inversa de la NAO.

5.2. Tecnologías para seguimiento del cuerpo humano

5.2.1. MediaPipe Holistic: predicción simultánea de rostros, manos y poses en el dispositivo

MediaPipe es un marco para la construcción de canalizaciones de Machine Learning aplicadas multimodales (por ejemplo, video, audio, cualquier serie temporal). MediaPipe ofrece soluciones de aprendizaje automático personalizables multiplataforma de código abierto para medios en vivo y de transmisión.

MediaPipe Holistic consiste en una nueva herramienta con componentes optimizados de pose, cara y mano que se ejecutan en tiempo real, con una transferencia de memoria mínima entre sus backends de inferencia y soporte adicional para la intercambiabilidad de los tres componentes, dependiendo de las compensaciones de calidad / velocidad. Al incluir los tres componentes, MediaPipe Holistic proporciona una topología unificada para más de 540 puntos clave innovadores (33 poses, 21 por mano y 468 puntos de referencia faciales) y logra un rendimiento casi en tiempo real en dispositivos móviles.También se puede trabajar con las API listas para usar de MediaPipe para investigación (Python) y web (JavaScript) para facilitar el acceso a la tecnología.

Para simplificar el traking de las poses del cuerpo humano se estará usando solamente la solución de MediaPipe Pose. Esta es una solución de aprendizaje automático para el seguimiento de posturas corporales de alta fidelidad, que infiere 33 puntos de referencia 3D y una máscara de segmentación de fondo en todo el cuerpo a partir de fotogramas de video RGB utilizando su investigación en BlazePose que también impulsa la API de detección de posturas del kit Machine Learnig. El modelo en MediaPipe Pose predice la ubicación de 33 Pose Landmarks como se muestra en la Figura 5.

5.3. Algoritmos para filtrado de las señales

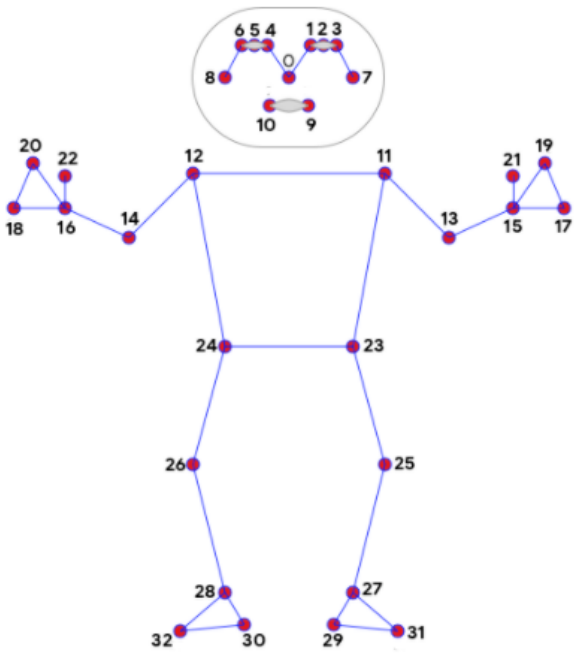
5.3.1. Filtro simplificado de Kalman

El filtro de Kalman es un algoritmo que se usa para medir variables no observables (como el ruido presente en las mediciones de las poses humanas) de un sistema dinámico lineal. En este caso sucede que no tenemos un modelo físico de los movimientos de las poses humanas que detecta el modelo de MediaPipe que queremos filtrar. Porque en realidad no sabemos como se movería la articulaciones de la persona una vez comience o termine el movimiento. En este difícil problema, podríamos poner toda la información desconocida del modelo físico en la variable aleatoria ξ_k:

$$x_{k+1} = x_k + \xi_k \tag{20}$$

Este tipo de sistema contiene la información desconocida para nosotros ya que no tenemos idea de la física del movimiento. No podemos decir que en diferentes momentos los errores son independientes entre sí y sus medias son iguales a cero. Este tipo de situaciones no

[7]



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figura 5: Pose landmarks

se aplica la teoría de Kalman. Pero de todos modos podemos intentar usar la maquinaria de la teoría de Kalman. Simplificando mas el problema se sabe que al aumentar el paso k, el coeficiente de Kalman siempre se estabiliza al valor determinado K_{stab}. Entonces, en lugar de calcular los valores para obtener el coeficiente de Kalman K_k mediante fórmulas difíciles, podemos suponer que este coeficiente es constante y seleccionar esa constante. Esta suposición no afectaría mucho al filtrado de resultados. Al final todo se vuelve muy simple, sin la necesidad de usar formulas de la teoría de Kalman, solo tenemos que seleccionar un valor razonable K_{stab} e insertarlo en la fórmula iterativa.

$$x_{k+1} = K_{stab} \cdot z_{k+1} + (1 - K_{stab}) \cdot x_k \tag{21}$$

5.3.2. Filtro de media móvil

El filtro de media móvil es un filtro FIR (respuesta de impulso finito) de paso bajo simple que se usa comúnmente para regular una matriz de datos / señales muestreados. Toma X muestras de entrada a la vez y toma el promedio de ellas para producir un solo punto de salida. A medida que aumenta la longitud del filtro, aumenta la suavidad de la salida. Este es otro metodo por el cual se pueden filtrar los LandkMarks de las poses obtenidas con MediaPipe Pose.

CAPÍTULO 6

Diseño

6.1. Mapeo de puntos en el espacio 3D al espacio de las articulaciones del robot NAO

6.1.1. Mapeo de las articulaciones de los brazos

Mapeo de vectores

Para utilizar las poses capturadas por MediaPipe para el robot humanoide, debemos transformarlo en el espacio del robot, ya que las posiciones están disponibles a partir de los datos de captura de movimiento de la cámara.Se usó la información vectorial para describir la relación relativa entre cada articulación en lugar de la información de ubicación.

$$ {}^{E}_{S}V_N = \frac{{}^{E}_{S}V_H}{||{}^{E}_{S}V_H||} * l_1 + \begin{bmatrix} 0 \\ l_0 \\ 0 \end{bmatrix} \tag{22} $$

$$ {}^{W}_{S}V_N = \frac{{}^{E}_{S}V_H}{||{}^{E}_{S}V_H||} * l_1 + \frac{{}^{W}_{E}V_H}{||{}^{W}_{E}V_H||} * l_2 + \begin{bmatrix} 0 \\ l_0 \\ 0 \end{bmatrix} \tag{23} $$

Cinemática directa de los brazos usando parámetros Denavit Hardenberg

La cadena cinemática de los brazos consiste en cuatro juntas.Se utilizan los parámetros de Denavit Hardenberg (D-H) para asignar un sistema de coordenadas ortogonales a la

derecha a cada eslabón de una cadena cinemática abierta.Las transformaciones entre las juntas adyacentes se pueden realizar mediante una matriz de transformación de coordenadas uniforme. Para asegurar la similitud de las trayectorias conjuntas se construyen dos cadenas de cinemáticas en cada brazo, de esta manera evitamos múltiples configuraciones al momento de posicionar el efector final en un punto 3d deseado. Primero, establecimos cadenas de trabajo desde el hombro hasta el codo en cada brazo del robot. El Cuadro 3 enumera parámetro DH de la primera cadena cinemática. Donde el valor de α esta dado en la ecuación 24.

$$\alpha = \tan^{-1} \frac{l_0}{l_1} \tag{24}$$

La segunda cadena cinemática se construye desde el hombro hasta la muñeca en los brazos del robot. El Cuadro 4 enumera los parámetros DH para la segunda cadena.

Cuadro 3: Parámetros de la cadena 1

Cadena	θ	d	a	A
0-1	$\theta_0 + \pi + \alpha$	0	0	$\frac{\pi}{2}$
1-2	$\theta_1 + \alpha$	0	$\sqrt{l_1^2 + l_0^2}$	$\frac{\pi}{2}$

Cuadro 4: Parámetros de la cadena 2

Cadena	θ	d	a	A
0-1	$\theta_0 + \pi$	0	0	$\frac{\pi}{2}$
1-2	$\theta_1 + \frac{\pi}{2}$	0	l_0	$\frac{\pi}{2}$
2-3	$\theta_2 + \pi$	l_1	0	$\frac{\pi}{2}$
3-4	$\theta_3 + \frac{\pi}{2}$	0	l_2	0

Los parámetros DH para las dos cadenas cinemáticas de la mano derecha son iguales a los de la mano izquierda, donde se trabaja con ángulos negativos en θ_1 y θ_2 debido a la orientación de la junta. La transformación DH que describe la traslación y la orientación de la articulación j basada en la articulación anterior j - 1 en el marco de referencia. La matriz DH general esta dada en la ecuación 25 donde $Rot(z, \theta_i)$ es una matriz de rotación que gira alrededor del eje z en θ_i y $Rot(x, A_i)$ es una matriz de rotación que gira alrededor del eje x con un ángulo A_i. $Trans(0, 0, d_i)$ es una matriz de traslación que se traduce a lo largo del eje z con d_i y $Trans(a_1, 0, 0)$ es una matriz de traslación que se traduce a lo largo del eje x con A_i.Entonces, la matriz de transformación para la cadena i se puede denotar usando la matriz DH como en la ecuación 26.

$$^{j}_{j-1}T = Rot(z, \theta_i) \cdot Trans(0, 0, d_i) \cdot Trans(a_1, 0, 0) \cdot Rot(x, A_i) \tag{25}$$

$$^{final}_{0}T = {}^{1}_{0}T \cdot {}^{2}_{1}T \cdots {}^{final}_{3}T \tag{26}$$

Por la primera cadena cinemática esta dada por la ecuación 27 dado el Cuadro 3 y la segunda cadena cinemática esta dada por la ecuación 28. Donde ${}_{j-1}^{j}T_{cd1}$ es la matriz de transformación de la matriz desde la junta j a la junta j-1 de la cadena 1 y ${}_{j-1}^{j}T_{cd2}$ es la matriz de transformación de la matriz desde la junta j a la junta j-1 de la cadena 2.

$$ {}_{S}^{E}V^{T} = {}_{0}^{1}T_{cd1} \cdot {}_{1}^{2}T_{cd1} \tag{27} $$

$$ {}_{S}^{W}V^{T} = {}_{0}^{1}T_{cd2} \cdot {}_{1}^{2}T_{cd2} \cdot {}_{2}^{3}T_{cd2} \cdot {}_{3}^{4}T_{cd2} \tag{28} $$

Solución cinemática inversa de los brazos

La cinemática directa puede encontrar el punto de un efector final en relación con con los valore dados de las articulaciones. Ahora nos centraremos en resolver el problema inverso. Encontrar el conjunto de valores para que las articulaciones conduzcan al efector final a un punto deseado en relación a la estructura el robot. La cinemática inversa que se presenta a continuación resuelve el problema de las dos cadenas cinemáticas propuestas para cada brazo del robot.

Nao se controlara utilizando las configuraciones de los ángulos de articulación, estos ángulos se calculan de acuerdo con los vectores 25 y 26 y las matrices de transformación 27 y 28. Para la cadena 1 en el brazo izquierdo, la matriz de transformación esta dada por ${}_{S}^{E}V^{T}$. Entonces, la posición del efector final en las coordenadas del hombro se puede obtener usando 30. En la cadena 1, la relación entre la posición de la junta final y los ángulos de las articulaciones esta representado por la ecuación 31.

Para la primera cadena cinemática sabemos que la cuarta columna de la matriz de transformación ${}_{S}^{E}V^{T}$ es equivalente al vector ${}_{S}^{E}V_{N}$ así como se muestra en la ecuación 29 y 30.

$$ {}_{S}^{E}V^{T}(:4) = {}_{0}^{1}T_{cd1} \cdot {}_{1}^{2}T_{cd1} \cdot [0\ 0\ 0\ 1]^{T} \tag{29} $$

$$ {}_{S}^{E}V^{T}(:4) = \begin{bmatrix} {}_{S}^{E}V_{N}(x) & {}_{S}^{E}V_{N}(y) & {}_{S}^{E}V_{N}(z) & 1 \end{bmatrix}^{T} \tag{30} $$

Donde ${}_{S}^{E}V_{N}(x)$ corresponde a la componente en x del vector ${}_{S}^{E}V_{N}$ y así respectivamente con los demás componentes (y) y (z). Al resolver para las matrices en las ecuaciones 29 y 30 se obtiene como resultado la ecuación 31 que contiene la relación entre los ángulos de la cadena cinemática y la posición final del efector.

$$ \begin{bmatrix} {}_{S}^{E}V_{N}(x) \\ {}_{S}^{E}V_{N}(y) \\ {}_{S}^{E}V_{N}(z) \\ 1 \end{bmatrix} = \begin{bmatrix} -\sqrt{l_0^2 + l_1^2}\cos(\theta_1 + \alpha)\cos\theta_0 \\ -\sqrt{l_0^2 + l_1^2}\cos(\theta_1 + \alpha)\sin\theta_0 \\ \sqrt{l_0^2 + l_1^2}\sin(\theta_1 + \alpha) \\ 1 \end{bmatrix} \tag{31} $$

Luego se puede despejar para cada uno de los ángulos obteniendo las ecuaciones 32, 33 y 34

$$\sin(\theta_1 + \alpha) = \frac{{}^{E}_{S}V_N(z)}{\sqrt{l_0^2 + l_1^2}} \tag{32}$$

$$\sin(\theta_0) = \frac{{}^{E}_{S}V_N(y)}{-\sqrt{l_0^2 + l_1^2}\cos(\theta_1 + \alpha)} \tag{33}$$

$$\cos(\theta_0) = \frac{{}^{E}_{S}V_N(x)}{-\sqrt{l_0^2 + l_1^2}\cos(\theta_1 + \alpha)} \tag{34}$$

De acuerdo con la ecuación 32 el ángulo calculado se encuentra dentro del rango de [-90,90] . Sin embargo, para las articulaciones del robot, el ángulo puede exceder o no entrar en el rango calculado. Por eso se debe realizar una interpolación para obtener un valor apropiado para el ángulo de la articulación según el Cuadro 2. Ya habiendo calculado el valor de "LShoulderRoll"dado por 32, es decir, θ_1 se puede obtener θ_0 el cual es el ángulo de la junta "LShoulderPitch" dado por la ecuación 34.

Para la cadena 2 en el brazo izquierdo, la matriz de transformación esta dada por ${}^{W}_{S}V^{T}$. Entonces, la posición del efector final en las coordenadas del hombro se puede obtener usando 36. En la cadena 2, la relación entre la posición de la junta final y los ángulos de las articulaciones esta representado por la ecuación 37.En la segunda cadena cinemática sabemos igualmente que la cuarta columna de la matriz de transformación ${}^{W}_{S}V^{T}$ es equivalente al vector ${}^{W}_{S}V_N$ así como se muestra en la ecuación 35 y 36.

$${}^{W}_{S}V^{T}(:4) = {}^{1}_{0}T_{cd2} \cdot {}^{2}_{1}T_{cd2} \cdot {}^{3}_{2}T_{cd2} \cdot {}^{4}_{3}T_{cd2} \cdot [0\ 0\ 0\ 1]^{T} \tag{35}$$

$${}^{W}_{S}V^{T}(:4) = \begin{bmatrix} {}^{W}_{S}V_N(x) & {}^{W}_{S}V_N(y) & {}^{W}_{S}V_N(z) & 1 \end{bmatrix}^{T} \tag{36}$$

Donde ${}^{W}_{S}V_N(x)$ corresponde a la componente en x del vector ${}^{W}_{S}V_N$ y así respectivamente con los demás componentes (y) y (z). Al resolver 35 y 36 se obtiene la ecuación 37.

$$\begin{bmatrix} {}^{W}_{S}V_N(x) \\ {}^{W}_{S}V_N(y) \\ {}^{W}_{S}V_N(z) \\ 1 \end{bmatrix} = \begin{bmatrix} -l_2 sin(\theta_3)(sin(\theta_0)sin(\theta_2) - sin(\theta_1)cos(\theta_0)cos(\theta_2)) + l_0 sin(\theta_1)cos(\theta_0) - \\ l_1 cos(\theta_0)cos(\theta_1) + l_2 cos(\theta_0)cos(\theta_1)cos(\theta_3) \\ -l_2 sin(\theta_3)(cos(\theta_0)sin(\theta_2) + sin(\theta_1)cos(\theta_2)sin(\theta_0)) + l_0 sin(\theta_1)sin(\theta_0) - \\ l_1 cos(\theta_1)sin(\theta_0) + l_2 cos(\theta_0)cos(\theta_1)cos(\theta_3) \\ l_0 - cos(\theta_1) - l_1 sin(\theta_1) - l_2 - sin(\theta_1)cos(\theta_3) + l_2 sin(\theta_3) - cos(\theta_2)cos(\theta_1) \\ 1 \end{bmatrix} \tag{37}$$

Ya teniendo la ecuación 37 a pesar de que ya se tienen los valores de θ_0 y θ_1, el sistema de ecuaciones obtenido no se puede resolver de manera analítica por lo que se tiene que sobre escribir la ecuación 35 de una manera mas amigable para poder despejar θ_2 y θ_3. De esta manera se mueven a la derecha de la ecuación todas las matrices de las cuales ya conocemos todos sus valores, obteniendo la ecuación 38.

$$ {}^{2}_{1}T^{-1}_{cd2} \cdot {}^{1}_{0}T^{-1}_{cd2} \cdot {}^{W}_{S}V^{T}(:4) = {}^{3}_{2}T_{cd2} \cdot {}^{4}_{3}T_{cd2} \cdot [0\ 0\ 0\ 1]^{T} \tag{38} $$

Resolviendo el lado izquierdo de las matrices de la ecuación 38 tenemos como resultado 39 y su ecuación simplificada en 40.

$$ \begin{bmatrix} {}^{W}_{S}X' \\ {}^{W}_{S}Y' \\ {}^{W}_{S}Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{{}^{W}_{S}V_N(z)cos(\theta_1)}{sin(\theta_1)^2+cos(\theta_1)^2} - l_0 + \frac{{}^{W}_{S}V_N(x)sin(\theta_1)cos(\theta_0))}{(sin(\theta_1)^2+cos(\theta_1)^2)(cos(\theta_0)^2+sin(\theta_0)^2)} + \\ \frac{{}^{W}_{S}V_N(y)sin(\theta_1)sin(\theta_0)}{(sin(\theta_1)^2+cos(\theta_1)^2)(cos(\theta_0)^2+sin(\theta_0)^2)} \\ \\ \frac{-{}^{W}_{S}V_N(x)sin(\theta_0)}{cos(\theta_0)^2+sin(\theta_0)^2} + \frac{{}^{W}_{S}V_N(y)cos(\theta_0)}{cos(\theta_0)^2+sin(\theta_0)^2)} \\ \\ \frac{-{}^{W}_{S}V_N(x)cos(\theta_0)cos(\theta_1)}{(cos(\theta_1)^2+sin(\theta_1)^2)(cos(\theta_0)^2+sin(\theta_0)^2))} + \frac{{}^{W}_{S}V_N(z)sin(\theta_1)}{cos(\theta_1)^2+sin(\theta_1)^2} - \\ \frac{{}^{W}_{S}V_N(y)cos(\theta_1)sin(\theta_0))}{((cos(\theta_1)^2+sin((\theta_1)^2)(cos(\theta_0)^2+sin(\theta_0)^2))} \\ \\ 1 \end{bmatrix} \tag{39} $$

$$ \begin{bmatrix} {}^{W}_{S}X' \\ {}^{W}_{S}Y' \\ {}^{W}_{S}Z' \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{W}_{S}V_N(x)sin(\theta_1)cos(\theta_0) + {}^{W}_{S}V_N(y)sin(\theta_1)sin(\theta_0) + {}^{W}_{S}V_N(z)cos(\theta_1) - l_0 \\ -{}^{W}_{S}V_N(x)sin(\theta_0) + {}^{W}_{S}V_N(y)cos(\theta_0) \\ -{}^{W}_{S}V_N(x)cos(\theta_0)cos(\theta_1) - {}^{W}_{S}V_N(y)cos(\theta_1)sin(\theta_0) + {}^{W}_{S}V_N(z)sin(\theta_1) \\ 1 \end{bmatrix} \tag{40} $$

Para facilitar el calculo, se resuelve el lado derecho de la ecuación 38 y el lado izquierdo puede ser reemplazado como se muestra en 41.

$$ {}^{2}_{1}T^{-1}_{cd2} \cdot {}^{1}_{0}T^{-1}_{cd2} \cdot {}^{W}_{S}V^{T}(:4) = \begin{bmatrix} {}^{W}_{S}X' \\ {}^{W}_{S}Y' \\ {}^{W}_{S}Z' \\ 1 \end{bmatrix} = \begin{bmatrix} l_2cos(\theta_2)sin(\theta_3) \\ l_2sin(\theta_2)sin(\theta_3) \\ l_2cos(\theta_3) + l_1 \\ 1 \end{bmatrix} \tag{41} $$

De esta manera se pueden obtener los ángulos θ_2 y θ_3 según las ecuaciones 42, 43 y 44.

$$ cos(\theta_3) = \frac{{}^{W}_{S}Z' - l_1}{l_2} \tag{42} $$

$$sin(\theta_2) = \frac{{}^{W}_{S}Y'}{l_2 sin(\theta_3)} \tag{43}$$

$$cos(\theta_2) = \frac{{}^{W}_{S}X'}{l_2 sin(\theta_3)} \tag{44}$$

Al igual que en la cadena 1, se tiene que tomar en cuenta el rango de movimiento de las articulaciones descrito en el Cuadro 2 para las juntas del robot, ya que el ángulo puede exceder o no entrar en el rango calculado. Por eso se debe realizar una interpolación para obtener un valor apropiado para el ángulo. Al final de todo el procedimiento se obtienen las ecuaciones que describen el mapeo desde el espacio 3D al espacio de las articulaciones del robot:

$$\theta_1 = \sin^{-1}\left(\frac{{}^{E}_{S}V_N(z)}{\sqrt{l_0^2 + l_1^2}}\right) + \alpha \tag{45}$$

$$\theta_0 = \cos^{-1}\left(\frac{{}^{E}_{S}V_N(x)}{-\sqrt{l_0^2 + l_1^2}\cos\left(\theta_1 + \alpha\right)}\right) \tag{46}$$

$$\theta_3 = \cos^{-1}\left(\frac{{}^{W}_{S}Z' - l_1}{l_2}\right) \tag{47}$$

$$\theta_2 = \cos^{-1}\left(\frac{{}^{W}_{S}X'}{l_2 sin(\theta_3)}\right) \tag{48}$$

Donde θ_1 es el ángulo en "LShoulderRoll", θ_0 es el ángulo en "ShoulderPitch", θ_3 es el ángulo en "LElbowRollz θ_2 es el ángulo en "LElbowYaw".

6.1.2. Mapeo de las articulaciones de la cabeza

Para las articulaciones de la cabeza se tienen que obtener los valores de los ángulos (HeadYaw y HeadPitch) debido a que el tracking de la cabeza no es muy consistente con las profundidades de los lankmarks, se decidió usar un método meramente geométrico para obtener los ángulos necesarios. Para estos se utilizan el vector formados entre los lankmars LEFTEAR y RiGHTEAR 5 y se mide su ángulo con respecto al eje x del robot, este sera el ángulo correspondiente a la articulación HeadYaw. Para la articulación HeadPitch se utiliza los vectores ya sean LEFTEAR y LEFTEYE5 o RIGHTEAR y RIGHTEYE5 y se mide su angulo con respecto al eje z del robot.

Para medir el ángulo entre los vectores se hace uso de la ecuación 49 y se realiza un mapeo de los ángulos de acuerdo a los rangos de las articulaciones de NAO mostrados en el Cuadro 1. En la Figura 6 se observan ejemplos del tracking de la cabeza y en la Figura 7 se puede observar los resultados de los lankmarks en un espacio 3D.

$$cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|} \tag{49}$$

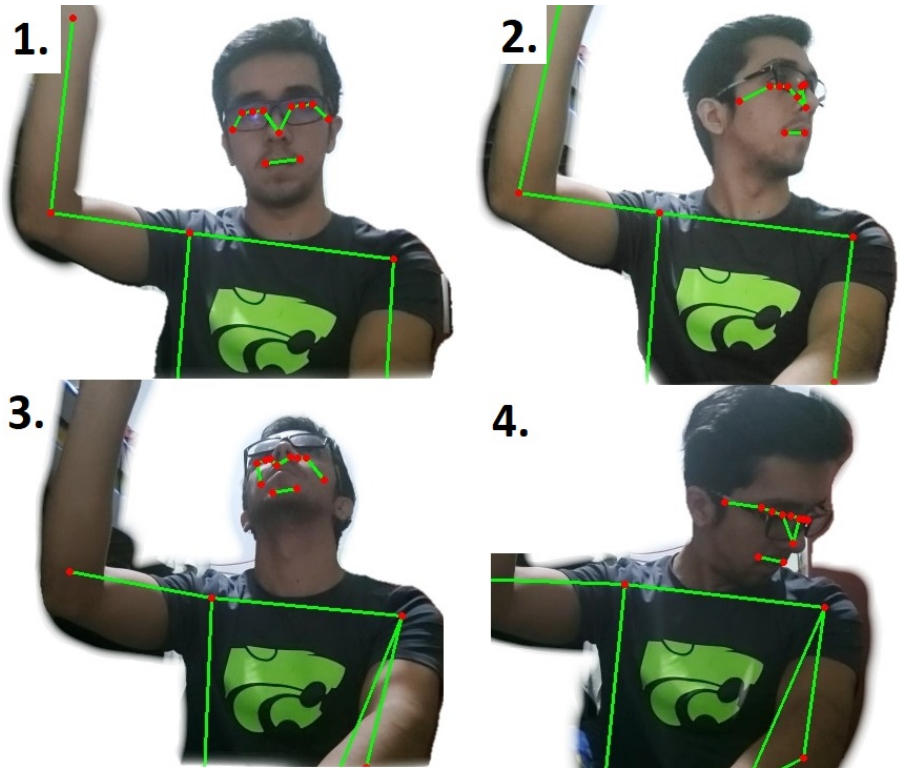


Figura 6: Poses de la cabeza captadas con cámara

6.2. Aplicación de filtro para suavizar los ángulos en las trayectorias

Para obtener trayectorias mas suaves se aplico un filtro simplificado de Kalman [21]. Este filtro se aplico debido a las variaciones que aveces representa el eje z en los lankmarks proporcionados por MediaPipe al captar las poses. Esto ayuda en su mayoría tener trayectorias mas suaves, esto también ayuda en posiciones estáticas ya que aveces se presenta una variación significativa en los ángulos obtenidos de las poses. En la Figura [8] se muestra la trayectoria del ángulo LShoulderPitch sin aplicar filtro y en la Figura [9] se observa la trayectoria del ángulo aplicando un filtro de Kalman simplificado [21] con una constante K=0.4.

6.3. Herramientas y lenguaje de programación utilizado

Se utilizaron dos scrips para el funcionamiento de NAO, uno escrito en Python 3.8 para el tranking de las poses y otro scrip escrito en Python 2.7 para el control del robot. Esto se debió a que el paquete de IDLE de NAO solamente es compatible con Python 2.7 y la

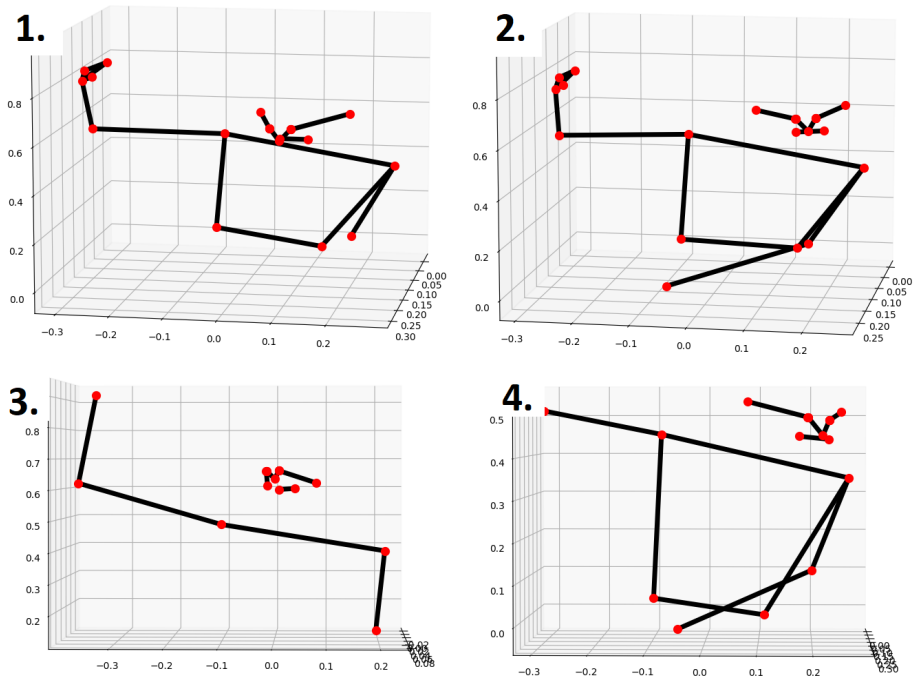


Figura 7: Poses de la cabeza en un plano 3D con los Lankmarks obtenidos

herramienta de MediaPipe solamente está disponible para Python 3.5 y superiores. Esto presenta un dificultad ya que se debe encontrar una manera para que ambos scrips estén en constante comunicación para compartir información de los ángulos obtenidos y del estado del robot.

Para resolver el problema de comunicación, se hacen uso de sockets para enviar mensajes a través de una red local. Estos proporcionan una forma de comunicación entre procesos (IPC).Para esto se propone el uso de programación de una conexión de dos nodos en una red para comunicarse entre sí. En esta caso, el script que controla el robot se configura como un socket que escucha un puerto en particular (cliente) en una IP local (ya que se está trabajando con una misma computadora). Mientras el script que detecta y envia las poses se configura como un socket (servidor) que forma el conector de escucha mientras el cliente se comunica con el servidor.

El uso de sockets también resuelve un problema de los procesos paralelos, ya que tanto el movimiento del robot como el proceso de tracking de poses, se ejecutan al mismo tiempo y si se programan esos procesos de manera serial, cada uno de ellos interrumpiría el proceso del otro lo que resultaría con que no se pudiera hacer el tracking de poses hasta que el robot termine su movimiento.

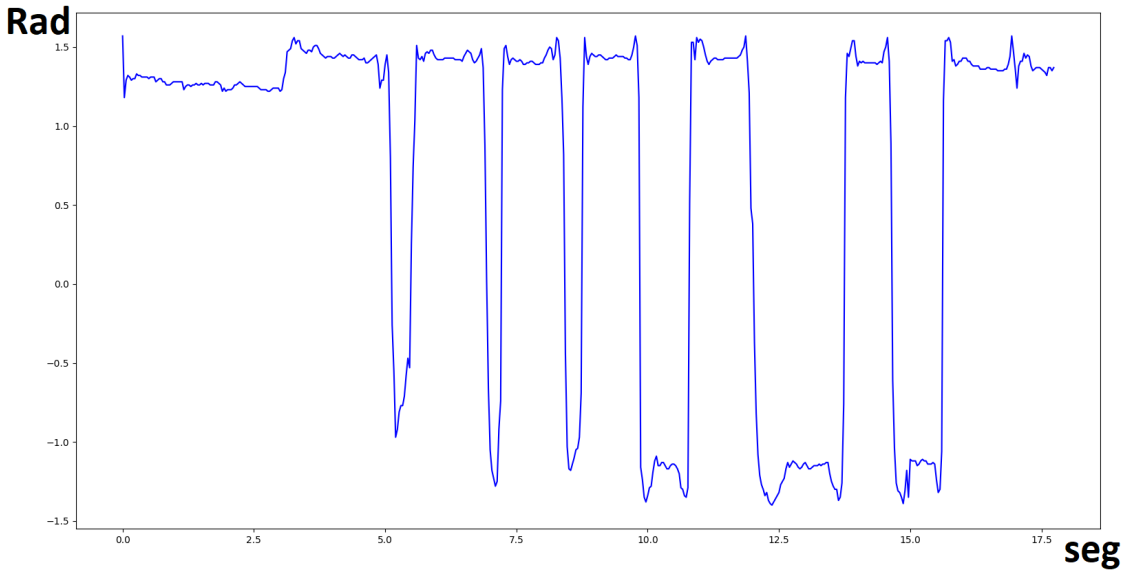


Figura 8: Trayectoria de un ángulo

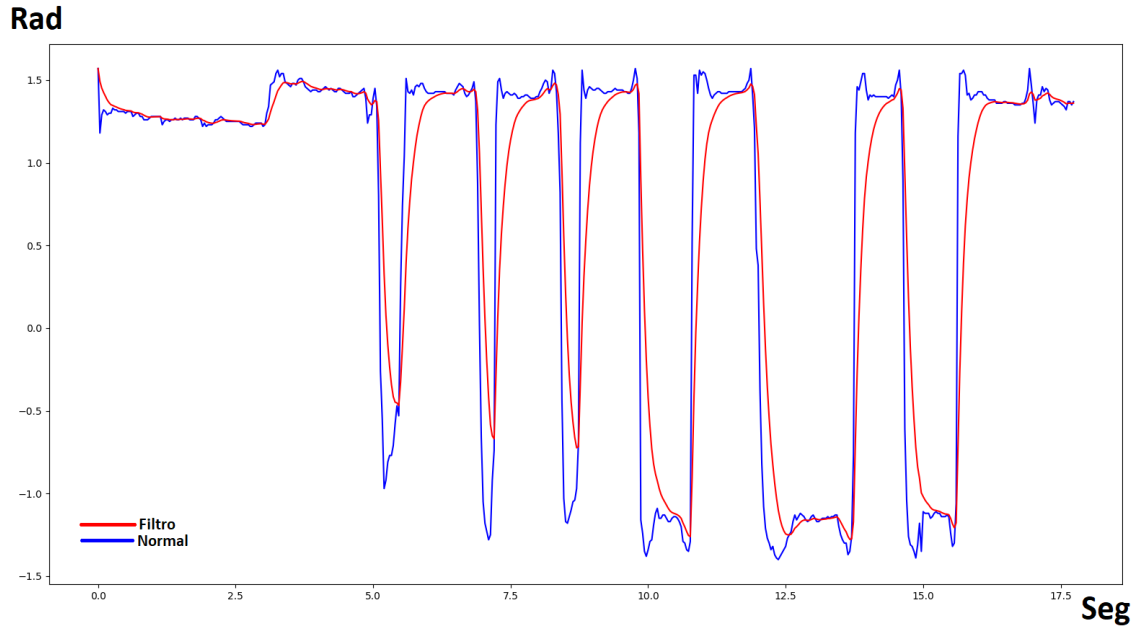


Figura 9: Trayectoria de un ángulo aplicando filtro simplificado de Kalman

CAPÍTULO 7

Resultados

7.1. Poses imitadas por NAO

Se realizaron distintas poses para comprobar el funcionamiento del trakinng e imitación de las juntas del robot. Se trabajaron solamente con las articulaciones del torso superior del robot, por lo que se excluyen las piernas. En la Figura 10 se comprobó la imitación de poses solamente con la cabeza, y en las Figuras 11 y 12 se comprueba la imitación de poses solamente con los brazos.

Se notó que se presenta dificultad al momento de imitar poses cuando los vectores en los brazos no son muy visibles y su magnitud se encuentra en su mayoría en el eje z (profundidad de la imagen) también se presenta dificultad al momento de realizar poses que están fuera del alcance del rango de las juntas del robot. Los ejemplos de estas dificultades se muestran en la Figura 13.

Las figuras presentan distintas poses haciendo uso de las articulaciones de la cabeza y ambos brazos para imitar las poses. Se puede notar que ha similitud en los ángulos de las poses mostradas.

7.1.1. Velocidad del algoritmo de imitación

La velocidad con que se calculan los ángulos (ya obteniendo la pose de MediaPipe), incluyendo el proceso de filtrado varia entre 1.1ms y 1.5ms. Estos tiempos son idéales ya que Mediapipe trabaja con un muestreo en la camara de 30fps, por lo que solamente se pueden obtener 30 poses por segundo. Y ya que nuestro proceso del cálculo de los ángulos es como máximo de 1.5ms, no hay retraso en la obtención de configuraciones de poses.

7.2. Trayectorias imitadas por NAO

También se realizó la imitación de las trayectorias, esto se hizo guardando cada pose obtenida durante un margen de tiempo estipulado, esto con la limitante de respetar las velocidades máximas de NAO, por lo que las imitaciones de las trayectorias en la mayoría de casos pueden ser mas lentas que el movimiento original de la persona a la cual se le está haciendo el tracking de sus poses. En las figuras 17 y 18 se pude observar la trayectoria de un ángulo, los puntos verdes en la grafica de la trayectoria representan los ángulos que el robot NAO va a imitar al momento de recrear la trayectoria. A su vez se puede guardar una trayectoria por medio de comando de voz para después hacer que el robot la replique de nuevo usando comando por voz.

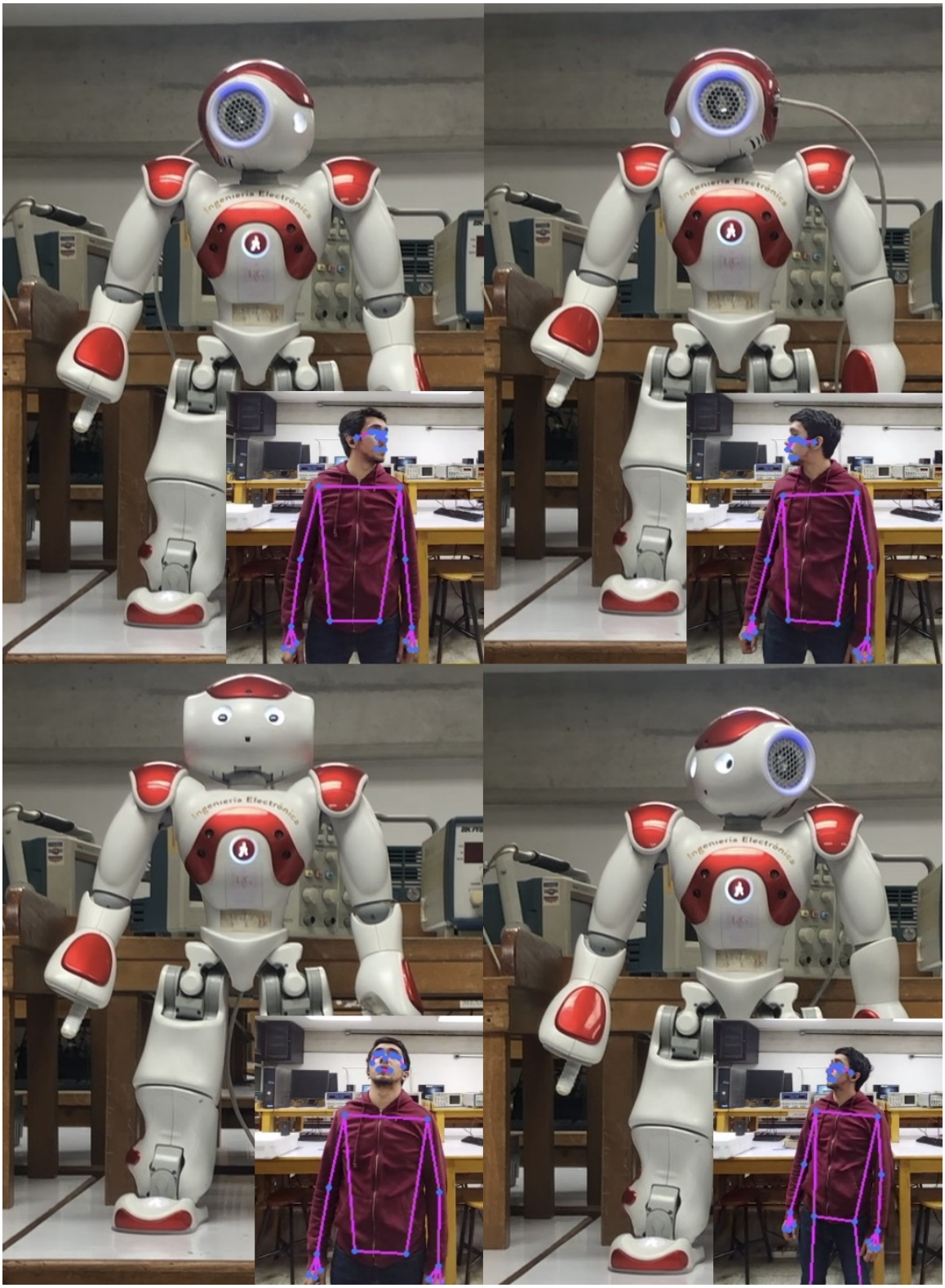


Figura 10: Poses imitando solamente de la cabeza

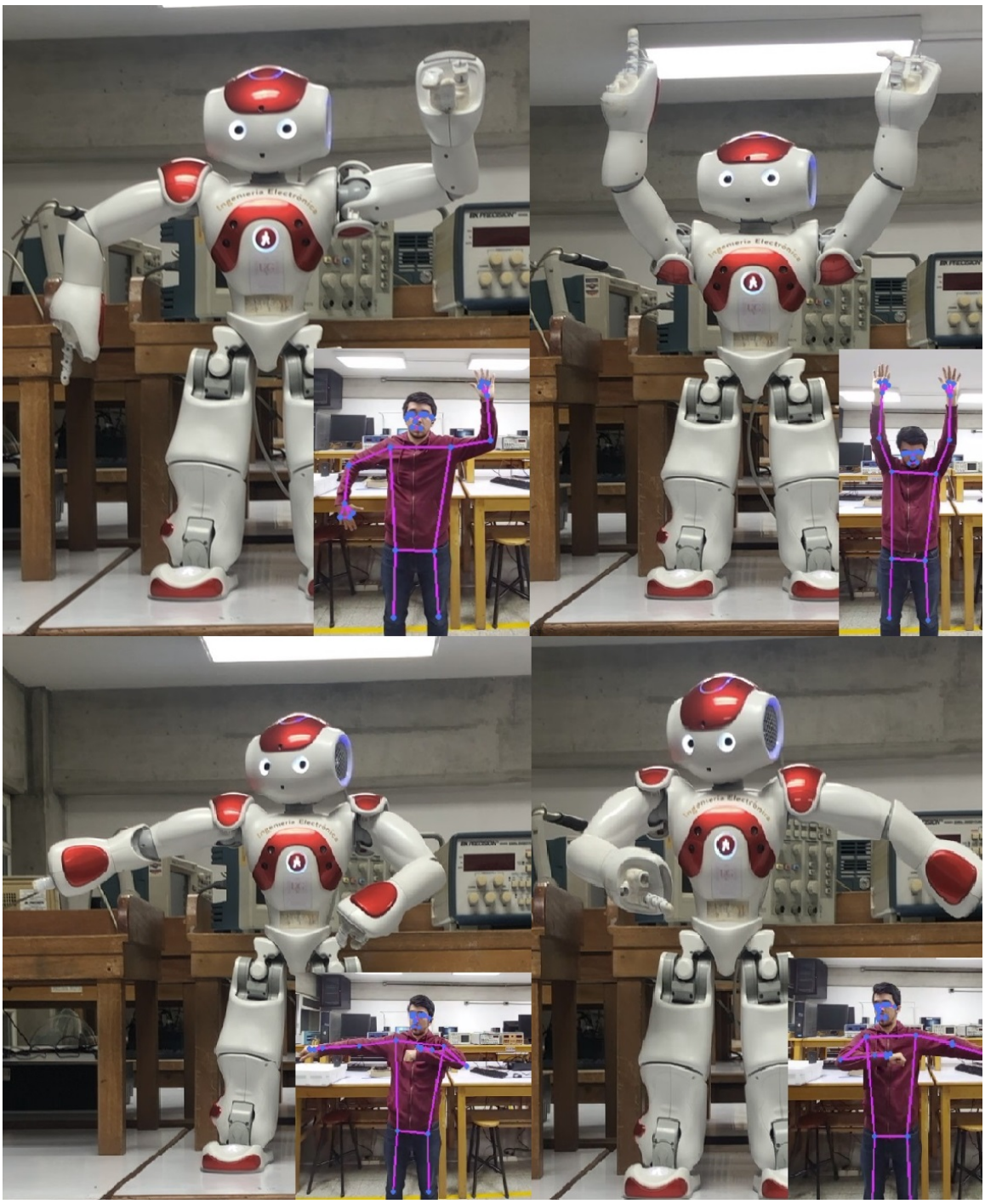


Figura 11: Poses imitando solamente los brazos 1

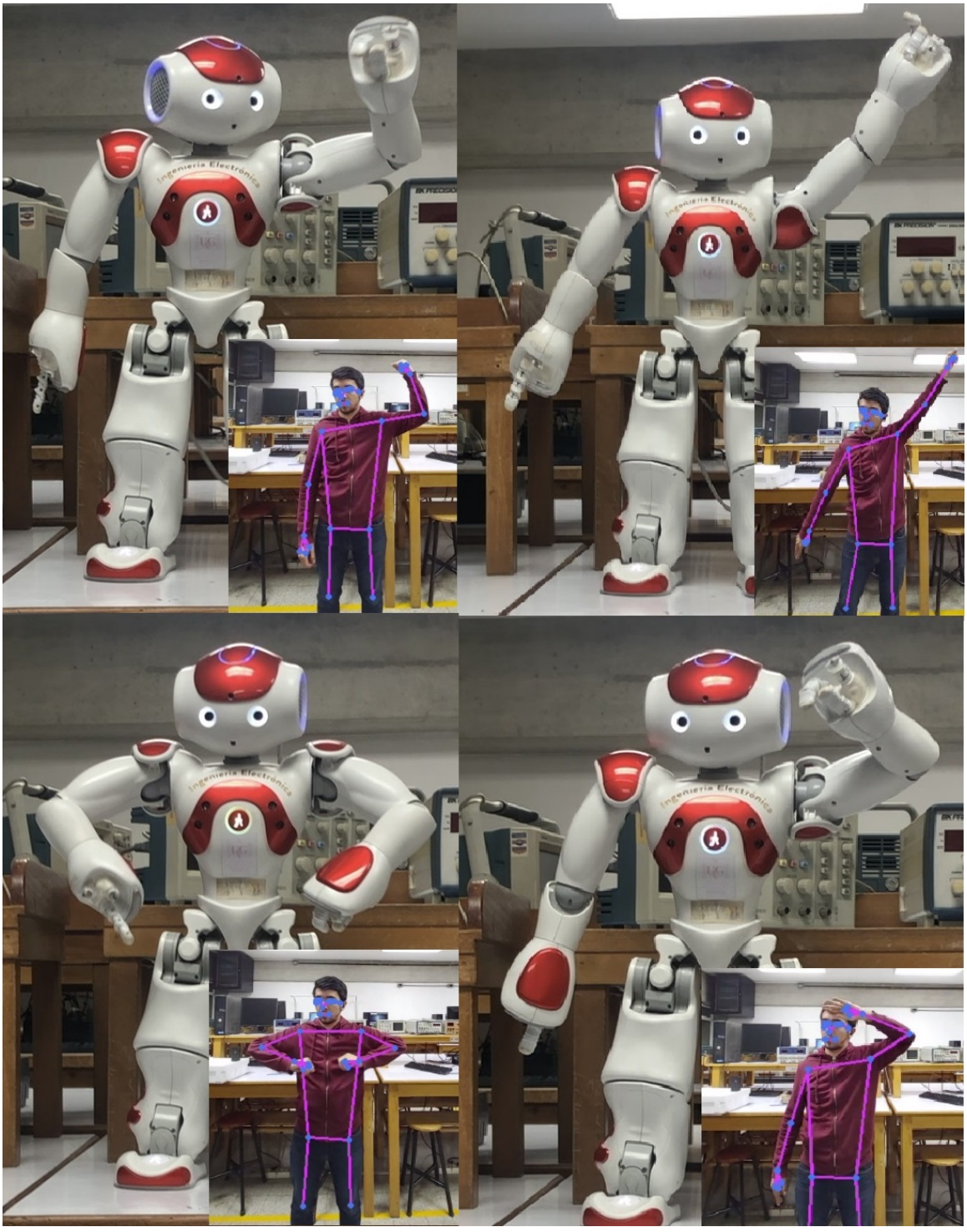


Figura 12: Poses imitando solamente los brazos 2

Figura 13: Imitación pobre de las poses

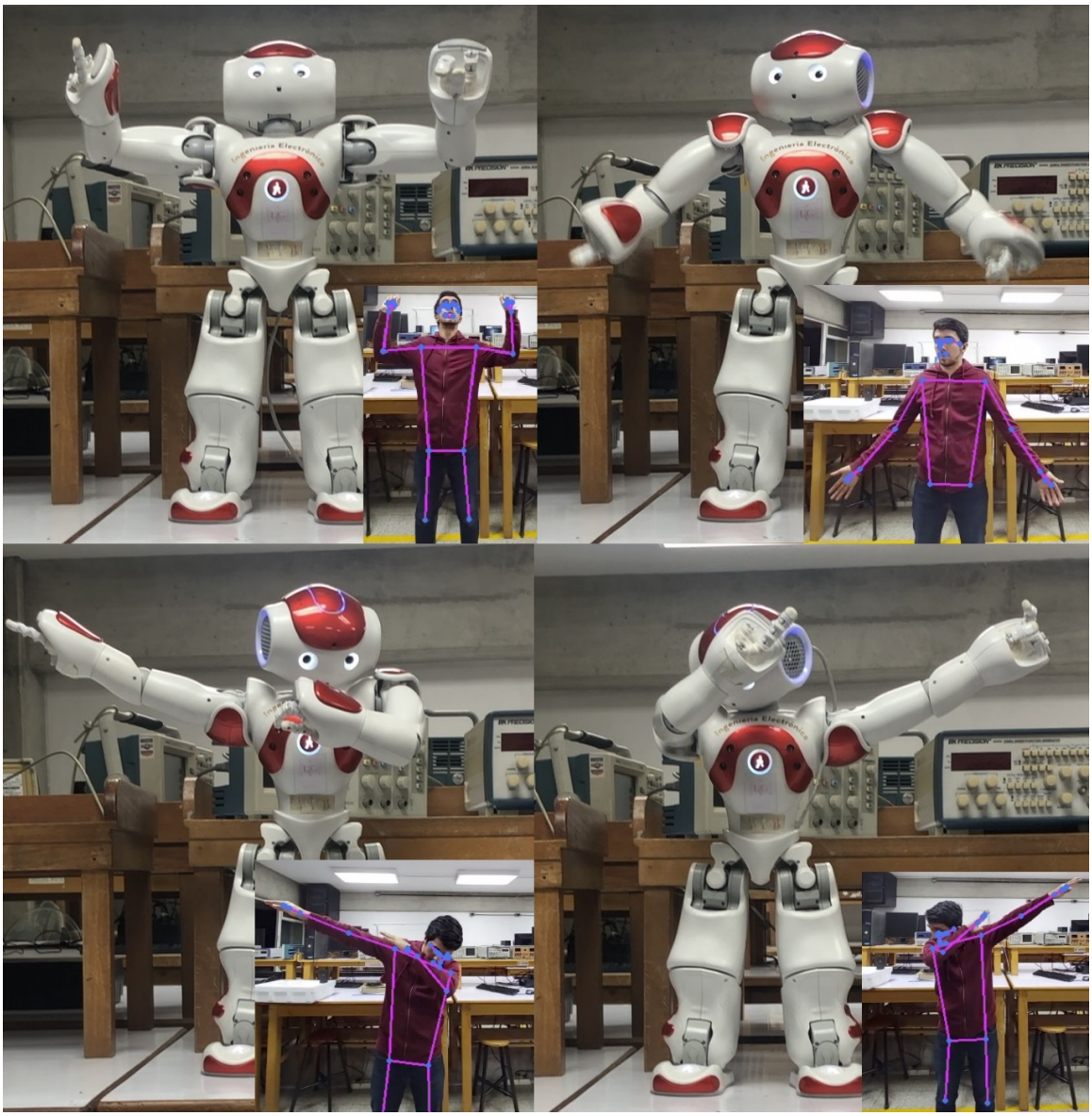


Figura 14: Poses imitando la cabeza y brazos 1

Figura 15: Poses imitando la cabeza y brazos 2

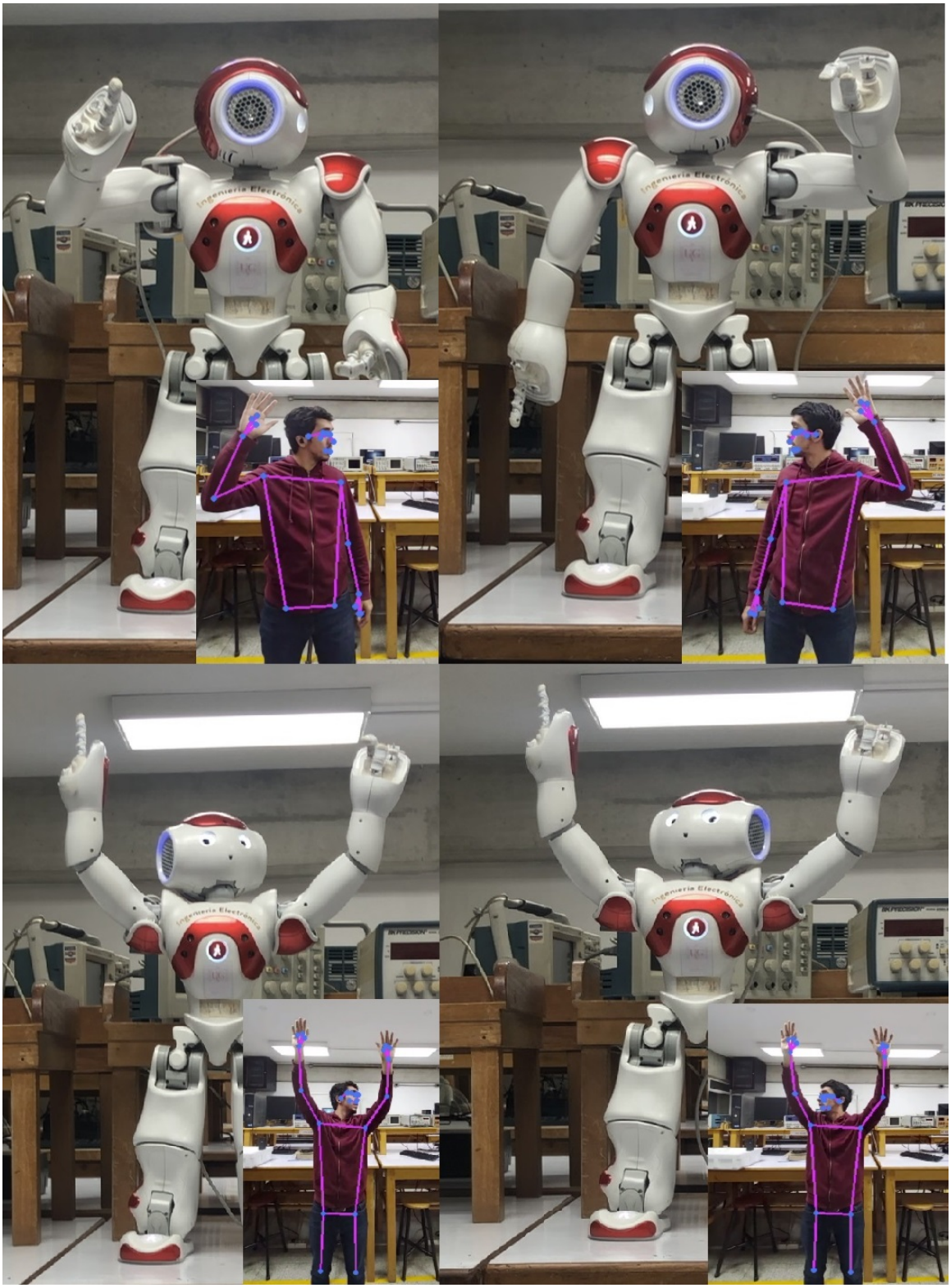


Figura 16: Poses imitando la cabeza y brazos 3

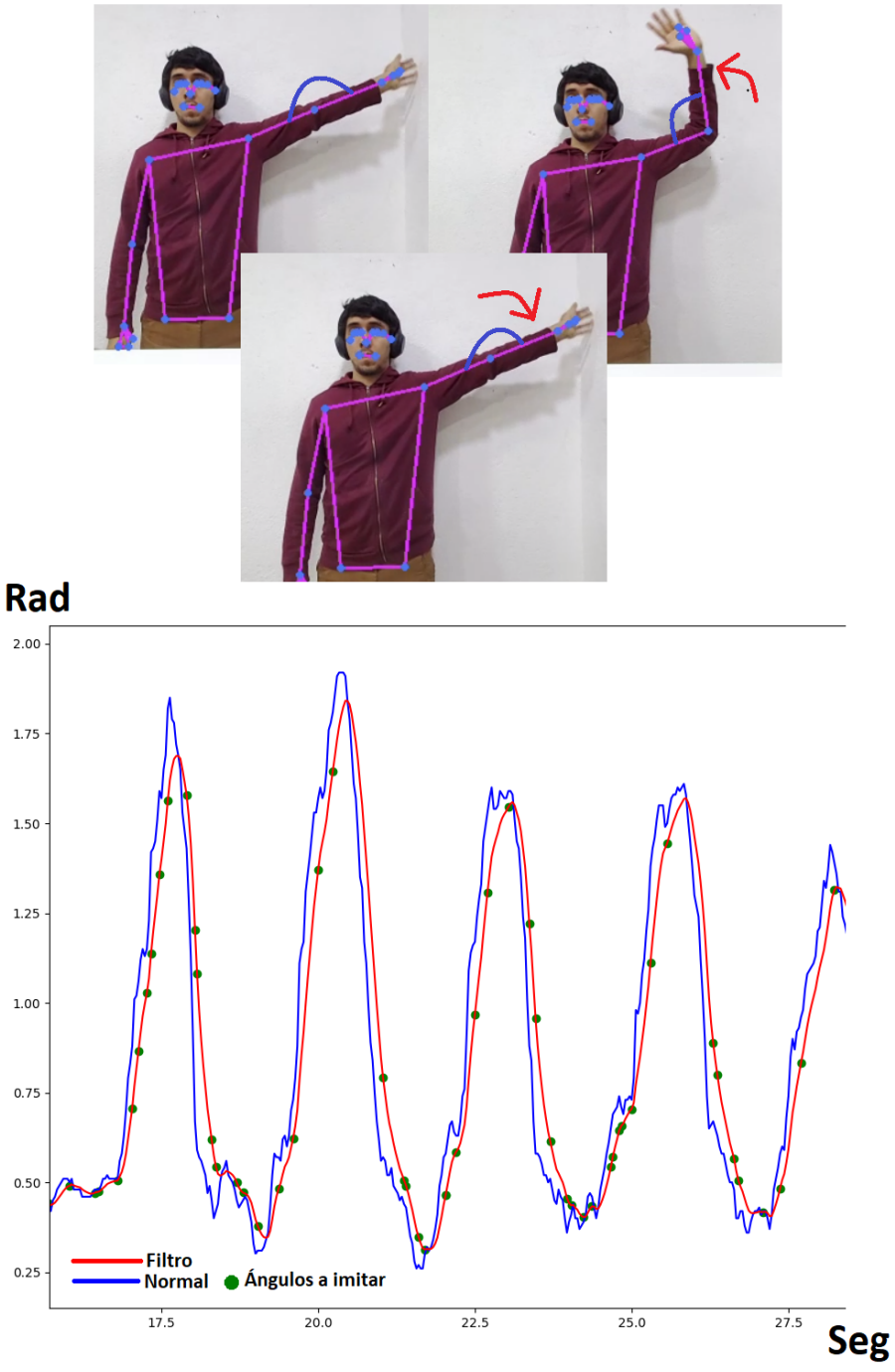


Figura 17: Trayectoria de un ángulo donde los puntos verdes en la gráfica de la trayectoria representan los ángulos que el robot NAO va a imitar al momento de recrear la trayectoria.

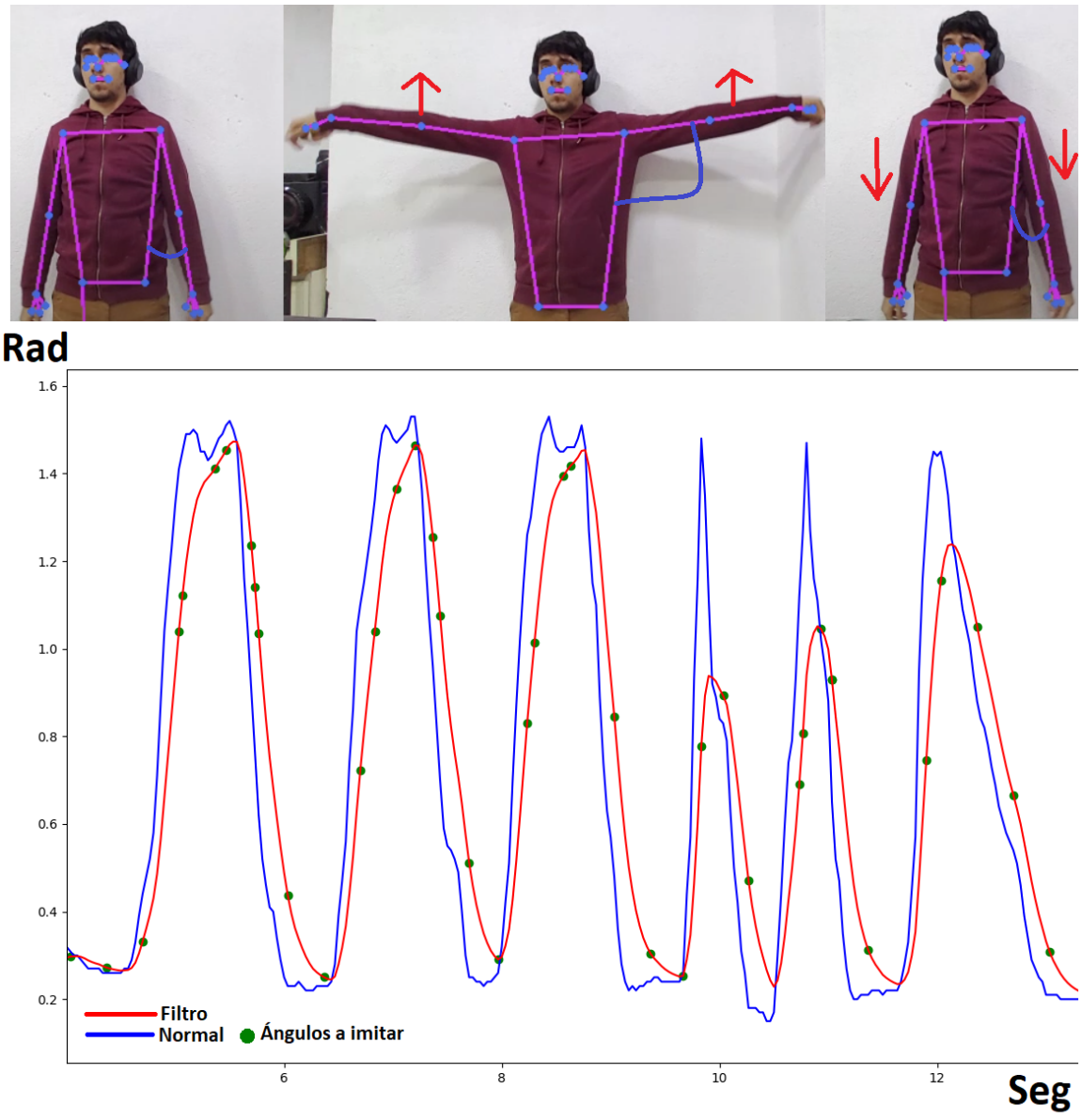


Figura 18: Trayectoria de un ángulo donde los puntos verdes en la gráfica de la trayectoria representan los ángulos que el robot NAO va a imitar al momento de recrear la trayectoria.

CAPÍTULO 8

Conclusiones

- Se diseñó un algoritmo que capta las poses del cuerpo humano y las puede traducir a las articulaciones del robot humanoide NAO, logrando que este las imite.
- Se logró el seguimiento satisfactorio del cuerpo humano mediante la herramienta de MediaPipe Pose.
- Se encontró una solución analítica para el problema de cinemática inversa de las juntas de los brazos de NAO, permitiendo un procesamiento mas rápido de los ángulos en tiempo real.
- Se encontró la cinemática directa para los brazos del robot NAO.
- Se diseñó una solución geométrica para realizar la imitación de los movimientos de la cabeza.
- Se diseñó un filtro para suavizas las curvas de los ángulos al momento de imitar trayectorias con el robot.
- Se diseñó un algoritmo para imitar las trayectorias de las poses en tiempo real.

CAPÍTULO 9

Recomendaciones

- Utilizar C++ para escribir el algoritmo de imitación de poses, ya que esto evitaría que se tuvieran que correr dos scrips distintos (uno en Ptyhon 2.7 y otro en Python 3.8).
- Utilizar las soluciones de MediaPipe Holistics para tener un mejor tracking de la posición y orientación de la cabeza, así como la orientación de las manos.
- Desarrollar un algoritmo para el equilibrio del centro de masa del robot, de esta manera también se podrían imitar las poses de la piernas.

CAPÍTULO 10

Bibliografía

[1] D. Z. Ming Zhang Jianxin Chen Xin Wei, «Work chain-based inverse kinematics of robot to imitate human motion with Kinect», *Wiley Etri*, pág. 11, 2018.

[2] N. Kofinas, «Forward and Inverse Kinematics for the NAO Humanoid Robot», Tesis doct., Technical University of Crete, Greece Department of Electronic y Computer Engineering, jun. de 2012.

[3] R. Jazar, *Theory of applied robotics : kinematics, dynamics, and control*. New York London: Springer, 2010, ISBN: 1441917497.

[4] C. III, *Kinematic Analysis of Robot Manipulators*. Cambridge: Cambridge University Press, 1998, ISBN: 978-0-521-04793-7.

[5] A. Robotics, *NAO*, [Web; accedido el 28-09-2021]. dirección: URL%7Bhttps://www.softbankrobotics.com/emea/es/nao%7D.

[6] ——, *Joints-NAO Software 1.14.5 documentation*, [Web; accedido el 12-08-2021]. dirección: URL%7Bhttp://doc.aldebaran.com/1-14/family/robots/joints_robot.html%7D.

[7] MediaPipe, *MediaPipe Pose*, [Web; accedido el 30-09-2021]. dirección: URL%7Bhttps://google.github.io/mediapipe/solutions/pose.html%7D.

CAPÍTULO 11

Anexos

11.1. Código algoritmo utilizado

Hay dos scrips para el funcionamiento de NAO, uno escrito en Python 3.8 para el tranking de las poses y otro scrip escrito en Python 2.7 para el control del robot.

Main.py (Python 3.7)

```
"""
Main scrip NAO control

@author: sergi
"""
#Socket server
import socket
import time
from planoAnguloHombro import should_angle2, should_angle,
calculate_angle, head_angle
import cv2
import mediapipe as mp
import numpy as np


HEADERSIZE = 5
#Initial angles if pose is noot detected
ListlangSho=list()
k=[1,1,1,1]
langSho =(1.57,0,0,0,1.57,0,0,0)
rangSho =(1.57,0,0,0,1.57,0,0,0)
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((socket.gethostname(), 1241))
s.listen(5)



mp_drawing = mp.solutions.drawing_utils
mp_pose = mp.solutions.pose

count=0

address=0
while (True):
    # now our endpoint knows about the OTHER endpoint.
    clientsocket, address = s.accept()
    print(f"Connection from {address} has been established.")
    if address!=0:
        break

cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.8, min_tracking_confidence=1)
as pose:
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(cv2.flip(frame, 1), cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Extract landmarks
        try:
            landmarks = results.pose_world_landmarks.landmark

            # Get coordinates
            shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,
            landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
            elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,
            landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
            wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,
```

```
landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]


relbow =      [landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].x,
landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].y,
landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value].z]
rshoulder = [landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].
landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].y,
landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].z]
rwrist =      [landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].x,
landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].y,
landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value].z]

lelbow =      [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,
landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y,
landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].z]
lshoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x
landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y,
landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].z]
lwrist =      [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,
landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y,
landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].z]
lear = [landmarks[mp_pose.PoseLandmark.LEFT_EAR.value].x,
landmarks[mp_pose.PoseLandmark.LEFT_EAR.value].y,
landmarks[mp_pose.PoseLandmark.LEFT_EAR.value].z]
rear = [landmarks[mp_pose.PoseLandmark.RIGHT_EAR.value].x,
landmarks[mp_pose.PoseLandmark.RIGHT_EAR.value].y,
landmarks[mp_pose.PoseLandmark.RIGHT_EAR.value].z]
leye = [landmarks[mp_pose.PoseLandmark.LEFT_EYE.value].x,
landmarks[mp_pose.PoseLandmark.LEFT_EYE.value].y,
landmarks[mp_pose.PoseLandmark.LEFT_EYE.value].z]
reye = [landmarks[mp_pose.PoseLandmark.RIGHT_EYE.value].x,
landmarks[mp_pose.PoseLandmark.RIGHT_EYE.value].y,
landmarks[mp_pose.PoseLandmark.RIGHT_EYE.value].z]

rhip= [landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].x,
landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].y,
landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].z]
lhip= [landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].x,
landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].y,
landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].z]
vHip=np.array(lhip)-np.array(rhip)
vSho=np.array(lshoulder)-np.array(rshoulder)
nvSho=np.sqrt(vSho.dot(vSho))
# Calculate angle
#print(vHip)
#print(np.sqrt(vHip.dot(vHip)))
#print(lshoulder,lelbow,lwrist)
```

```
            angle = calculate_angle(shoulder, elbow, wrist)
            langSho=should_angle2(lshoulder,lelbow,lwrist,1)
            rangSho=should_angle2(rshoulder,relbow,rwrist,-1)
            hangle=head_angle(lshoulder,rshoulder,lear,rear,leye,reye)
            angSho2=should_angle(lshoulder,lelbow)

            #ListlangSho.append(langSho)


            #print(lshoulder,lelbow,angSho)
            print(rangSho)
            print(langSho)
            print(hangle)
            #print(rangSho)
            #print(angSho2)
            #print(int(angle))
            print("     ")
            #print(int(angle))


            # Visualize angle
            cv2.putText(image, str(rangSho)+str(langSho)+str(int(hangle))+
            str(int(angle)),
                        tuple(np.multiply([0.01,0.05],
                        [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                        (255, 255, 255), 2, cv2.LINE_AA
                             )
            cv2.putText(image, str(vSho)+str(nvSho),
                        tuple(np.multiply([0.01,0.10],
                        [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                        (255, 255, 255), 2, cv2.LINE_AA
                             )

        except:
            pass


        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks,
        mp_pose.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66),
                                  thickness=2,
                                  circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(245,66,230),
                                  thickness=2, circle_radius=2))
```

```
            count=count+1
            #time.sleep(0.1)
            cv2.imshow('Mediapipe Feed', image)
            if count>0:
                msg = str(langSho)+", "+str(rangSho)+", "+str(hangle)
                #msg = f"{len(msg):<{HEADERSIZE}}"+msg
                msg = f"{'x':<{HEADERSIZE}}"+msg
                clientsocket.send(bytes(msg,"utf-8"))
                print (msg)
                count=0
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break

    cap.release()
    cv2.destroyAllWindows()
```

```
PlanoAnguloHombro.py (Python 3.7)
"""
Created on Mon Jul 9 19:37:12 2021

@author: sergi
"""
import numpy as np
from sympy import sin, cos, pi

c1=[16.204,14.853,3.683]
a=[5,17,5]
b=[0.737,10.775,11.563]
c=[-3.683,16.204,14.853];
c1=[3.683,16.204,14.853]
a=[0.5692636370658875, 0.0719519779086113, -0.4102078080177307]
b=[0.5426207184791565, 0.30124425888061523, -0.48975610733032227]
c=[0.3791402280330658, 0.43326127529144287, -0.5057594776153564]

a1=[5,5,17]
b1=[11.563,0.737,10.775]
```

```
def make_interpolater(value,min1,max1,min2,max2):
    range1=max1-min1
    range2=max2-min2
    # Convert the range1 into a 0-1 range (float)
    valueScaled = float(value - min1) / float(range1)
    value=min2 + (valueScaled * range2)
    if value>max2:
        value=max2
    if value<min2:
        value=min2
    return value

def calculate_angle(a,b,c):
    a = np.array(a) # First
    b = np.array(b) # Mid
    c = np.array(c) # End

    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) -
    np.arctan2(a[1]-b[1], a[0]-b[0])
    angle = np.abs(radians*180.0/np.pi)

    if angle >180.0:
        angle = 360-angle

    return angle

def should_angle(a,b):
    a = np.array(a) # Origen
    b = np.array(b) # Final
    v1=(b[2]-a[2],b[1]-a[1],b[0]-a[0])

    vz0=(v1[0],v1[1],0)
    vx0=(0,v1[1],v1[2])
    vzy0=(0,v1[1],0)

    angle0 = np.arccos(np.dot(vz0, vzy0) /
    (np.linalg.norm(vz0) * np.linalg.norm(vzy0)))*180/np.pi
    angle1 = np.arccos(np.dot(vx0, vzy0) /
    (np.linalg.norm(vx0) * np.linalg.norm(vzy0)))*180/np.pi

    #if v1[1]<0:
    #    angle0=90-angle0+90

    return int(angle0),int(angle1)

def Rot(ang,axe):
    if axe==1:
        rotM=[[1,0,0,0],[0,cos(ang),-sin(ang),0],[0,sin(ang),cos(ang),0],
```

```
        [0,0,0,1]]
    if axe==2:
        rotM=[[cos(ang),0,sin(ang),0],[0,1,0,0],[-sin(ang),0,cos(ang),0],
        [0,0,0,1]]
    if axe==3:
        rotM=[[cos(ang),-sin(ang),0,0],[sin(ang),cos(ang),0,0],[0,0,1,0],
        [0,0,0,1]]

    return rotM

def Trans(p):
    transV=[[1,0,0,p[0]],[0,1,0,p[1]],[0,0,1,p[2]],[0,0,0,1]]
    return transV

def head_angle(a,b,c,d,e,f):
    a = np.array(a)
    b = np.array(b)
    c = np.array(c)
    d = np.array(d)
    e = np.array(e)
    f = np.array(f)

    Vba=[a[0]-b[0],a[2]-b[2]]
    Vdc=[c[0]-d[0],c[2]-d[2]]
    V0dc=[Vdc[0],0]

    Vce=e-c
    Vdf=f-d
    V0ce=Vce*[1,0,1]
    V0df=Vdf*[1,0,1]




    ang0=np.arccos(np.dot(Vba, Vdc) / (np.linalg.norm(Vba) *
    np.linalg.norm(Vdc)))*180/np.pi
    ang1=np.arccos(np.dot(V0dc, Vdc) / (np.linalg.norm(V0dc) *
    np.linalg.norm(Vdc)))*180/np.pi
    ang2=np.arccos(np.dot(Vce, V0ce) / (np.linalg.norm(Vce) *
    np.linalg.norm(V0ce)))*180/np.pi
    ang3=np.arccos(np.dot(Vdf, V0df) / (np.linalg.norm(Vdf) *
    np.linalg.norm(V0df)))*180/np.pi
    if(Vdc[1]<0):
        ang0=-ang0
        ang1=-ang1
    if(Vce[1]<0):
        ang2=-ang2
    if(Vdf[1]<0):
```

```
            ang3=-ang3
        if(ang1>0):
            ang2=ang3


        ang1=make_interpolater(int(ang1),-45,35,-90,90)
        ang4=int(make_interpolater(ang2,-8,10,-4,4))
        ang4=int(make_interpolater(ang4,-4,4,-32,25))

        return round(ang1/180*pi,2),round(ang4/180*pi,2)

#Funcion that gets de angles of the joints of the robots arms
def should_angle2(a,b,c,s):
    #Definition of NAO constant joints
    l0=0
    #l1=0.105
    #l2=0.05595
    #x=np.arctan(l0/l1)
    idx = [1,2,0]
    a = np.array(a) # Shoulder
    b = np.array(b) # Elbow
    c = np.array(c) # Wrist
    a=a[idx]
    b=b[idx]
    c=c[idx]

    esVh=b-a
    weVh=c-b
    wsVh=c-a
    #NesVh
    l1=np.sqrt(esVh.dot(esVh))
    l2=np.sqrt(weVh.dot(weVh))
    #print(esVh,weVh)
    #esVh[1]=0
    #weVh[1]=0
    #l1=np.sqrt(esVh.dot(esVh))
    #l2=np.sqrt(weVh.dot(weVh))
    #print(l1,l2)
    l1=0.21

    try:
        esVh[1]=-np.sqrt(-esVh[0]**2-esVh[2]**2+(l1)**2)
        weVh[1]=-np.sqrt(-weVh[0]**2-weVh[2]**2+(l1*0.9)**2)
    except:
        print("error")
    if(np.isnan(weVh[1])==True):
        #weVh=c-b
        weVh[1]=0
```

```
    if(np.isnan(esVh[1])==True):
        esVh=b-a

    #NweVh

    l2=np.sqrt(weVh.dot(weVh))
    l1=np.sqrt(esVh.dot(esVh))
    #print(esVh,weVh)
    wsVh=esVh+weVh

    ang1=(np.arcsin(esVh[2]/l1))
    ang0=np.arccos(esVh[0]/(-(l1)*np.cos(ang1)))
    ang0y=np.arcsin(esVh[1]/(-(l1)*np.cos(ang1)))
    #if ang0y<0:
    #    ang0=-ang0

    xii=wsVh[0]*np.sin(ang1)*np.cos(ang0)
    +wsVh[1]*np.sin(ang1)*np.sin(ang0)+wsVh[2]*np.cos(ang1)
    yii=-wsVh[0]*np.sin(ang0) + wsVh[1]*np.cos(ang0)
    zii=-wsVh[0]*np.cos(ang0)*np.cos(ang1)-
    wsVh[1]*np.cos(ang1)*np.sin(ang0)+wsVh[2]*np.sin(ang1);

    ang3=np.arccos((zii-l1)/l2)
    #ang33=(np.arccos(weVh.dot(esVh)/(l1*l2)))*180/np.pi
    ang2y=np.arccos(xii/(l2*np.sin(ang3)))
    ang2=np.arccos(yii/(l2*np.sin(ang3)))

    #if ang2y<0:
    #    ang2=-ang2
    if(np.isnan(ang2)==True):
        ang2=0

    try:
        ang3=float(ang3)

    except:
        print("NA")


    return  round(ang0-np.pi/2,2),
    round(ang1*s,2),round(ang2-np.pi/2,2),round(ang3,2)

def should_angle3(a,b,c):
    #Definition of NAO constant joints
    l0=0.015
    l1=0.105
    l2=0.05595
    x=np.arctan(l0/l1)
```

```
    a = np.array(a) # Shoulder
    b = np.array(b) # Elbow
    c = np.array(c) # Wrist

    esVh=b-a
    weVh=c-b

    esVn= (esVh/np.sqrt(esVh.dot(esVh)))*l1+[0,l0,0]
    wsVn= (esVh/np.sqrt(esVh.dot(esVh)))*l1+[0,l0,0]+
    (weVh/np.sqrt(weVh.dot(weVh)))*l2

    ang1=np.arcsin(esVn[0]/np.sqrt((l0**2+l1**2)))
    ang0=-np.arccos(esVn[1]/((-np.sqrt((l0**2+l1**2)))*np.cos(ang1+x)))
    ang0y=np.arcsin(esVn[2]/((-np.sqrt((l0**2+l1**2)))*np.cos(ang1+x)))
    ang2=2
    ang3=3
    #ang3=np.arccos(((wsVn[1]*np.cos(ang0)*np.cos(ang1) +
    wsVn[0]*np.sin(ang1) - wsVn[2]*np.cos(ang1)*np.sin(ang0))-l1)/l2)
    #ang2=np.arccos((wsVn[0]*np.cos(ang1) - l0 +
    wsVn[1]*np.sin(ang1)*np.cos(ang0)+
    wsVn[2]*np.sin(ang1)*np.sin(ang0))/(l2*np.sin(ang3)))

    return int(ang0*180/np.pi),int(ang1*180/np.pi),
    int(ang2*180/np.pi),int(ang3*180/np.pi)

#print (should_angle2(a, b,c,1))
#print( calculate_angle(a,b,c))
#print (should_angle3(a, b, c))
print(should_angle(a,b))
```

ClientNAOJoints.py (python 2.7)

```
import socket
# Choregraphe simplified export in Python.
from naoqi import ALProxy
import math
import time
names = list()
times = list()
keys = list()

HEADERSIZE = 5
```

```
trigger=True
bSize=200

angs11=[1.57,0,0,0,1.57,0,0,0]

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((socket.gethostname(), 1241))
angs1=[1.57,0,0,0,1.57,0,0,0]

#Funcion para convertir el string con los angulos del robot a una lista
def Convert1(string):
    string=string.replace('(', '').replace(')', '')
    li = list(string.split(", "))
    return li
def Convert0(string):
    li = list(string.split("x    "))
    le=list()
    for i in li:
        le.append(Convert1(i))
        print i
    return le


while True:
    full_msg = ''
    new_msg = True
    while True:
        while trigger==True:
            msg = s.recv(bSize)
            print len(msg)

            full_msg += msg.decode("utf-8")
            if len(msg)<bSize:
                mlist=(Convert0(full_msg))
                trigger=False
        full_msg = ""
        #time.sleep(1)
        angs1=mlist[len(mlist)-1]
        angs0=mlist[1]
        dif=0
        for i in range(len(angs1[1])-2):
            dif=abs(float(angs1[i])-float(angs11[i]))+dif

        angs11=angs1
        print dif
        print angs0
        print angs1
        full_msg = ""
```

```
trigger=True
if abs(dif)>0.05:
    names.append("LShoulderPitch")
    times.append([1.5])
    keys.append(float(angs1[0]))

    names.append("LShoulderRoll")
    times.append([1])
    keys.append(float(angs1[1]))

    names.append("LElbowYaw")
    times.append([1.5])
    keys.append(float(angs1[2]))

    names.append("LElbowRoll")
    times.append([1])
    keys.append(-float(angs1[3]))


    names.append("RShoulderPitch")
    times.append([1.5])
    keys.append(float(angs1[4]))

    names.append("RShoulderRoll")
    times.append([1])
    keys.append(-float(angs1[5]))

    names.append("RElbowYaw")
    times.append([1.5])
    keys.append(-float(angs1[6]))

    names.append("RElbowRoll")
    times.append([1])
    keys.append(float(angs1[7]))


names.append("HeadYaw")
times.append([0.7])
keys.append(float(angs1[8]))

names.append("HeadPitch")
times.append([0.5])
keys.append(float(angs1[9]))

try:
    # uncomment the following line and modify the IP
    #if you use this script outside Choregraphe.
    motion = ALProxy("ALMotion", "169.254.235.8", 9559)
```

```
        motion.angleInterpolation(names, keys, times, True)
        print("lol")
        names = list()
        times = list()
        keys = list()
    except BaseException, err:
        print err
```