TELEOPERACIÓN DE UN ROBOT HUMANOIDE *ROBOPHILO* A TRAVÉS DE UN SENSOR *KINECT*

AUTORES:
MIGUEL ÁNGEL AGUILAR LIZCANO
JHONATHAN ZAMBRANO ZAMBRANO

FACULTAD DE INGENIERÍA
PONTIFICIA UNIVERSIDAD JAVERIANA
BOGOTÁ, D.C.
2015

TELEOPERACIÓN DE UN ROBOT HUMANOIDE *ROBOPHILO* A TRAVÉS DE UN SENSOR *KINECT*

MIGUEL ÁNGEL AGUILAR LIZCANO
JHONATHAN ZAMBRANO ZAMBRANO

Trabajo de grado presentado para obtener el título de
Ingeniero electrónico

DIRECTOR

JULIÁN COLORADO MONTAÑO, Ph.D
Ingeniero Electrónico

FACULTAD DE INGENIERIA
PONTIFICIA UNIVERSIDAD JAVERIANA
BOGOTÁ, D.C.
2015

ARTICULO 23 DE LA RESOLUCION NO 13 DE JUNIO DE 1946

"La universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se duplique nada contrario al dogma y la moral católica y porque los trabajos no contengan ataques o polémicas puramente personales. Antes bien, que se vea en ellos el anhelo de buscar la verdad y la justicia".

AGRADECIMIENTOS

Al finalizar un trabajo tan arduo y lleno de dificultades como el desarrollo de una tesis es inevitable que te asalte un muy humano egocentrismo que te lleva a concentrar la mayor parte del mérito en el aporte que has hecho. Sin embargo, el análisis objetivo te muestra inmediatamente que la magnitud de ese aporte hubiese sido imposible sin la participación de personas e instituciones que han facilitado las cosas para que este trabajo llegue a un feliz término. Por ello, es para nosotros un verdadero placer utilizar este espacio para ser justo y consecuente con ellas, expresándoles nuestros agradecimientos.

En este presente trabajo de grado y en el desarrollo del mismo queremos rendir un especial agradecimiento a nuestras familias por su apoyo incondicional a lo largo de nuestras carreras, a nuestros compañeros y particularmente al ingeniero JULIÁN COLORADO MONTAÑO por su acompañamiento, paciencia y apoyo tanto personal como profesional. Agradecemos igualmente la Pontificia Universidad Javeriana por darnos la oportunidad de terminar esta etapa de nuestras vidas y por último y no menos importante a Dios por permitirnos vivir esta experiencia y por ayudar a terminarla de la mejor manera.

Tabla de contenido

1. INTRODUCCIÓN

Este proyecto nace del avance que ha tenido la industria del entretenimiento en las últimas décadas, observando el progreso tecnológico en la interacción de los seres humanos con los videojuegos, la industria de los videojuegos ha cambiado radicalmente en casi todos sus aspectos con el paso de los años, y un claro ejemplo es la complejidad de sus mandos. Si vamos un paso más adelante encontramos una nueva forma de interacción como el *WiiMote* o el *PlayStation Move* que a pesar de estar creados inicialmente para el servicio de videoconsolas, cada vez son más utilizados para comunicación con un ordenador mediante el movimiento corporal de una persona.

El objetivo de este proyecto nace de la necesidad de utilizar diversas tecnologías ya existentes y enfocarlas en dinamizar la interacción de los seres humanos con plataformas virtuales de entretenimiento pasando a un ámbito más atractivo y activo. Además la rama de robótica en la Pontificia Universidad Javeriana no cuenta con un gran conocimiento e investigación con humanoides y podrá servir para generar un avance en un ámbito que está en constante exploración a nivel mundial.

La realización de este proyecto permitirá que los usuarios interactúen con un objeto físico, en nuestro caso un robot humanoide. Logrando un avance en el control de plataformas de entrenamiento, además podría expandirse la utilidad del control de humanoides ayudando otros sectores como lo son: la industria para la realización de actividades que sobrepasen la capacidad física del ser humano como por ejemplo levantar cargas muy pesadas entre otros, en el ámbito institucional, medicina para ayudar a las personas que tengan problemas de coordinación motora(motricidad), militares permitiendo que no sean seres humanos los que se enfrentan sino humanoides controlados por los soldados.

La evolución de los mandos llega al punto de hacer el mejor acoplamiento del ser humano con los nuevos videojuegos, de allí nace *Kinect* como una herramienta que mejora la interacción entre los usuarios y los videojuegos, debido al éxito que tiene el sensor visual RGB-D Kinect, avanzo del ámbito de los videojuegos a ser un factor importante en la educación, innovación e investigación permitiendo que las aplicaciones del sensor se estén incrementado día a día.

Se quiere implementar el proceso académico llevado hasta el momento, donde este trabajo de grado nos permite partir de la teoría para luego llegar a la práctica, como sustenta Mansard (2012) que determina la efectividad de utilizar los procesos de control ya establecidos sin alterar la teoría para llevarlos a su ejecución mediante conjuntos de tareas arbitrarias con prioridades para conjuntos jerárquicos de tareas. Debido a lo anterior utilizaremos las leyes básicas de control permitiéndonos empezar desde la teoría en su diseño y luego pasar a simulaciones y finalmente pruebas en un robot físico [1].

Se encontró el estudio realizado por Kofman (2005) sobre la teleoperación de robots mediante interfaces visuales explica que la teleoperación de un humanoide mediante un manipulador remoto es a menudo necesaria en entornos dinámicos no estructurados cuando la presencia del ser humano en el sitio no es necesaria. Basado en lo anterior podemos decir que para impulsar el control de plataformas robóticas mediante sensores visuales que le permitan al usuario comunicarse en varios grados de libertad al tiempo, es necesaria la retroalimentación que pueda tener el robot al control, permitiendo un funcionamiento más exacto además en trabajos futuros pueda ser diseñado para tareas de mayor complejidad [2].

Debido a la investigación de Salgado (1994) donde determinan la viabilidad de realizar un control tipo PD para un robot de dos grados de libertad partiendo de un punto de equilibrio único. De acuerdo con lo

anterior se optó por realizar un control tipo PD para el funcionamiento del humanoide, debido a que es más sencillo obtener las ganancias proporcionales y derivativas partiendo de un punto de equilibrio único como sucede en nuestro caso [3].

2. MARCO TEÓRICO

Para la realización de este proyecto fue necesario repasar los conocimientos adquiridos sobre los modelos cinemáticos aplicados a la robótica, entendiendo como base el funcionamiento de un brazo robótico para luego aplicarlo en el humanoide *Robophilo*.

En este trabajo de grado se utilizó el sensor RGB-D Kinect principalmente diseñada para la videoconsola *XBOX 360,* debido a que era necesaria la interacción de la plataforma con un computador, fue necesaria la instalación de un software distribuido gratuitamente por *Microsoft* llamado "*Kinect for Windows"*, además de controladores necesarios para el reconocimiento de la plataforma por el computador.

La comunicación entre el humanoide y el computador se realizó por medio del protocolo de comunicación RS-232, además para permitir la comunicación entre el humanoide y el usuario, fue necesario adquirir paquete de desarrollo de software (SDK), suministrado por la empresa *RoboBrothers, inc.*

2.1. CAMARAS RGB-D KINECT

Figura 2.1.1. Sensor visual RGB-D Kinect [5].

Uso cámaras RGB-D

Originalmente conocido por el nombre en clave "Project Natal" , es un controlador de juego libre y entretenimiento creado por Alex Kipman, desarrollado por Microsoft para la videoconsola Xbox 360, y desde junio del 2011 para PC a través de Windows 7 y Windows 8. El sensor visual RGB-D Kinect permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce gestos, comandos de voz y objetos e imágenes.

El sensor visual RGB-D Kinect, normalmente, es usado para percepción en 3D de los movimientos humanos, sin embargo, sus usos pueden ir más allá, pues también se puede utilizar para aplicaciones robóticas. De entrada se ha liberado código abierto en LibFreenect para Linux, Microsoft mismo ha lanzado el sensor visual RGB-D Kinect SDK que básicamente usa Visual Studio 2012, y por su parte Google lanzó Robotic Operating System (ROS).

Especificaciones:

Sensores

- Lentes de color y sensación de profundidad
- Micrófono multi-arreglo
- Ajuste de sensor con su motor de inclinación
- Totalmente compatible con las consolas existentes de Xbox 360

Campo de visión

- Campo de visión horizontal: 57 grados
- Campo de visión vertical: 43 grados
- rango de inclinación física: ± 27 grados
- Rango de profundidad del sensor: 1,2 - 3,5 metros

Flujo de datos

- 320 × 240 a 16 bits de profundidad @ 30fps
- 640 × 480 32-bit de color @30fps
- Audio de 16-bit @ 16 kHz

Sistema de Seguimiento

- Rastrea hasta 6 personas, incluyendo 2 jugadores activos
- Rastrea 20 articulaciones por jugador activo
- Capacidad para mapear jugadores activos en Live Avatars

Sistema de audio

- Chat en vivo y voz dentro del juego (requiere Xbox Live Gold)
- Sistema de cancelación de eco que aumenta la entrada de voz
- Reconocimiento de voz múltiple

Funcionamiento

Este consiste en proyectar un patrón de luz conocido y a partir de un sensor de imagen calcular la deformación de este patrón sobre la superficie a ser analizada. En el caso el sensor visual RGB-D Kinect, el emisor de patrón de luz estructurada consta de 3 elementos. Primero un láser puntual, segundo un difusor que genera un patrón pseudo aleatorio y tercero un emisor óptico difractivo (DOE por sus siglas en inglés), este último tiene la propiedad de producir diferentes puntos de enfoque a diferentes distancias lo que lleva a que los puntos producidos por la emisión láser tengan en conjunto una textura diferente al ser proyectados sobre objetos a diferentes distancias [4], [5].

Esqueleto en el sensor visual RGB-D Kinect

Una de las muchas funciones que puede cumplir a cabalidad el sensor visual RGB-D Kinect es el *Skeleton Tracking*, es el nombre que se le da por hacer el seguimiento del cuerpo (esqueleto) de un usuario en tiempo real, debido a esto, Microsoft anuncio la comercialización libre de un SDK propio para poder utilizar el *Skeleton Tracking* con el sensor visual RGB-D Kinect. Lo que llevo al surgimiento de un movimiento para crear drivers y aplicaciones para el sensor visual RGB-D Kinect de código libre. A partir

de esto se creó Open NI (Open Natural Interaction), una organización para promover la compatibilidad de dispositivos.

Open NI creo una librería para utilizar el *Skeleton Tracking* incluida en un paquete llamado NITE y esta librería nos permite seguir el cuerpo del usuario aportando las coordenadas en el espacio de cada punto del cuerpo de interés tanto de extremidades como de articulaciones.

En esta parte desarrollamos un código en MATLAB que nos permitiera reconocer nuestros movimientos implementando el sensor el sensor visual RGB-D Kinect, para esto miramos cuantos grados de liberta queremos analizar y que datos son relevantes en cada caso, para hacer una mejor explicación lo ilustramos en la siguiente gráfica:

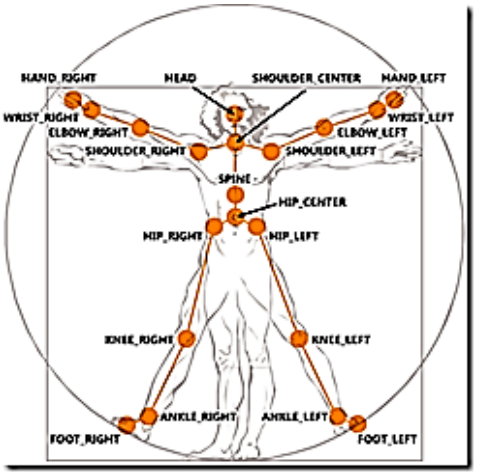


Figura 2.1.3. Articulaciones identificadas por el sensor visual el sensor visual RGB-D Kinect.

En nuestro caso vamos a tomar los siguientes puntos:

- Hombro izquierdo (Shoulder Left).
- Hombro derecho (Shoulder Right).
- Codo izquierdo (Elbow Left).
- Codo derecho (Elbow Right).
- Cadera izquierda (Hip Left).
- Cadera derecha (Hip Right).
- Rodilla izquierda (Knee Left).
- Rodilla derecha (Knee Right).
- Tobillo izquierdo (Ankle Left).
- Tobillo derecho (Ankle Right).
- Pie izquierdo (Foot Left).
- Pie derecho (Foot Right).

2.1.1. MODELO CINEMÁTICO DIRECTO

Un robot humanoide puede ser analizado como varias cadenas cinemáticas de cuerpos rígidos acoplados unos con otros a través de articulaciones rotacionales. A estos sistemas se les conoce como robots ramificados. Por tanto, el problema cinemático directo, tiene como objeto encontrar la posición cartesiana

de los últimos cuerpos que componen dichas cadenas en función de las variables articulares (ángulo de cada articulación). Por ejemplo, un brazo es una cadena cinemática compuesta normalmente por 6 grados de libertad: 2 grados en el hombro, 1 grado en el codo y 3 grados en la muñeca. El problema cinemático directo permite hallar la posición cartesiana de la mano (último cuerpo de la cadena) en función de las posiciones articulares previamente detalladas.

Nos permite a partir de un conjunto de parámetros físicos, que definen la geometría de un manipulador y de los ángulos articulares hallar la posición y orientación del efector final en un espacio tridimensional. Por medio de los parámetros (Denavit-Hartenberg) D—H [9], matrices de transformación homogénea y la obtención del algoritmo de cinemática directa, podemos llegar a obtener el modelo cinemático directo del humanoide.

Estos son los pasos del algoritmo genérico para la obtención de los parámetros D-H que se detallan a continuación [8]:

a) **Numerar los eslabones**: Se numerará como eslabón "0" a la base fija del robot, luego numerar los eslabones comenzando con "1" (el primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil de la cadena), n es la cantidad de grados de libertad de la cadena.

b) **Numerar las articulaciones**: numerar cada articulación comenzando con "1" que será el primer grado de libertad, y "n" el último.

c) **Localizar el eje de cada articulación**: si es rotativa el eje será su propio eje de giro. Si es prismática será el eje a lo largo del cual se produce el desplazamiento.

d) **Ejes Z**: Empezamos a colocar los sistemas de coordenadas XYZ. Situamos los Z_{i-1} en los ejes de las articulaciones i, con i=1,…, n. Es decir, Z_0 va sobre el eje de la 1ª articulación, Z_1 va sobre el eje del 2º grado de libertad, etc.

e) **Sistema de coordenadas 0**: Se sitúa el punto origen en cualquier punto a lo largo de Z_0. La orientación de X_0 e Y_0 puede ser arbitraria, siempre que se respete evidentemente que XYZ sea un sistema dextrógiro.

f) **Resto de sistemas**: Para el resto de sistemas i=1,…,n-1, colocar el punto origen en la intersección de Z_i con la normal común a Z_i y Z_{i+1}. En caso de cortarse los dos ejes Z, colocarlo en ese punto de corte. En caso de ser paralelos, colocarlo en algún punto de la articulación i+1.

g) **Ejes X**: Cada X_i va en la dirección de la normal común a Z_{i-1} y Z_i, en la dirección de Z_{i-1} hacia Z_i.

h) **Ejes Y**: Una vez situados los ejes Z y X, los Y tienen su direcciones determinadas por la restricción de formar un XYZ dextrógiro.

i) **Sistema del extremo del robot**: El n-ésimo sistema XYZ se coloca en el extremo del robot (herramienta), con su eje Z paralelo a Z_{n-1} y X_n sea normal a Z_{n-1} y Z_n.

La matriz D-H de un robot, se construye a partir de 4 parámetros geométricos:

Ángulos teta: Cada θ_i es el ángulo desde X_{i-n} hasta X_i girando alrededor de Z_{i-1}.
Distancias d: Cada d_i es la distancia desde el sistema XYZ i-1 hasta la intersección de las normales común de Z_{i-1} hacia Z_i, a lo largo de Z_{i-1}.
Distancias a: Cada a_i es la distancia de separación desde la intersección del eje Z_{i-1}con el eje X_i hasta el origen del sistema i-ésimo a lo largo del eje X_i (o la distancia más corta entre los ejes Z_{i-1} y Z_i cuando los ejes de articulacion son paralelos).
Ángulos alfa: Ángulo que hay que rotar Z_{i-1} para llegar a Z_i, rotando alrededor de X_i.

2.1.2. MODELO CINEMÁTICO INVERSO

Nos permite a partir de un conjunto de parámetros físicos que definen la geometría de un manipulador y de una posición y orientación específica del efector final se halla el conjunto de ángulos que dan como resultado dicha posición y orientación el efector final.

El jacobiano tiene traslaciones y rotaciones diferenciales; se pueden expresar de la siguiente manera

$$T + dT = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\partial_z & \partial_y & 0 \\ \partial_z & 1 & -\partial_x & 0 \\ -\partial_y & \partial_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} T$$

Diferencial de traslación

Diferencial rotación (matriz skew-simétrica)

Donde puede estar expresado como:

$$dT = \Delta T = \begin{bmatrix} 0 & -\partial_z & \partial_y & d_x \\ \partial_z & 0 & -\partial_x & d_y \\ -\partial_y & \partial_x & 0 & d_z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Donde $\partial = (\partial_x, \partial_y, \partial_z)^T$ es la rotación diferencial alrededor de los ejes principales del sistema de coordenadas de la base y $d = (d_x, d_y, d_z)^T$ es la translación diferencial a lo largo de los ejes principales del mismo sistema *k*.

Para representar las transformaciones en términos del sistema de coordenadas en movimiento, *m*, se post-multiplica por Δ y los elementos de translación y rotación se miden con respecto al sistema de coordenadas *T*. Esto se puede expresar de la siguiente manera,

$$dT = T({}^m\Delta)$$

$${}^m\Delta = T^{-1}dT = T^{-1}\Delta T$$

Las transformaciones diferenciales desarrolladas anteriormente expresan la posición y orientación del efector final del manipulador en términos del producto de transformaciones *A*.

$$T_6 = A_1 A_2 A_3 A_4 A_5 A_6$$

Se puede escribir una ecuación para representar un cambio diferencial dT_6 en términos de un cambio diferencial en coordenadas articulares, como:

$$^{m}\Delta = T^{-1} dT = T^{-1} \Delta T$$

$$dT_6 = T_6 {}^{T6}\Delta_i dq_i = A_1 A_2 \ldots A_{i-1} {}^{i-1}\Delta A_i A_{i+1} \ldots A_6 dq_i$$

Finalmente la solución cinemática inversa para un manipulador de 6 grados de libertad está dada por:

$$\begin{matrix} \frac{\partial T_n}{\partial q_1} & \frac{\partial T_n}{\partial q_2} & \frac{\partial T_n}{\partial q_3} & \frac{\partial T_n}{\partial q_4} & \frac{\partial T_n}{\partial q_5} & \frac{\partial T_n}{\partial q_6} \end{matrix} \leftarrow \text{Columnas del Jacobiano}$$

$$\begin{bmatrix} T_{6_{d_x}} \\ T_{6_{d_y}} \\ T_{6_{d_z}} \\ T_{6_{\partial_x}} \\ T_{6_{\partial_y}} \\ T_{6_{\partial_z}} \end{bmatrix} = \begin{bmatrix} T_{6_{d_{1_x}}} & T_{6_{d_{2_x}}} & T_{6_{d_{3_x}}} & T_{6_{d_{4_x}}} & T_{6_{d_{5_x}}} & T_{6_{d_{6_x}}} \\ T_{6_{d_{1_y}}} & T_{6_{d_{2_y}}} & T_{6_{d_{3_y}}} & T_{6_{d_{4_y}}} & T_{6_{d_{5_y}}} & T_{6_{d_{6_y}}} \\ T_{6_{d_{1_z}}} & T_{6_{d_{2_z}}} & T_{6_{d_{3_z}}} & T_{6_{d_{4_z}}} & T_{6_{d_{5_z}}} & T_{6_{d_{6_z}}} \\ T_{6_{d_{1_x}}} & T_{6_{d_{2_x}}} & T_{6_{d_{3_x}}} & T_{6_{d_{4_x}}} & T_{6_{d_{5_x}}} & T_{6_{d_{6_x}}} \\ T_{6_{d_{1_y}}} & T_{6_{d_{2_y}}} & T_{6_{d_{3_y}}} & T_{6_{d_{4_y}}} & T_{6_{d_{5_y}}} & T_{6_{d_{6_y}}} \\ T_{6_{d_{1_z}}} & T_{6_{d_{2_z}}} & T_{6_{d_{3_z}}} & T_{6_{d_{4_z}}} & T_{6_{d_{5_z}}} & T_{6_{d_{6_z}}} \end{bmatrix} \begin{bmatrix} dq_1 \\ dq_2 \\ dq_3 \\ dq_4 \\ dq_5 \\ dq_6 \end{bmatrix}$$

Para llegar al modelo cinemático inverso necesitamos obtener el jacobiano resultante de cada uno de los grados de libertad, es decir cada algoritmo de cinemática inversa existe un jacobiano. El algoritmo de cinemática inversa se obtiene de manera muy similar al directo.

Entonces el jacobiano resulta como una matriz de 6xn, donde n es el número de grados de libertad del robot:

$$T_n^0 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde las columnas del jacobiano son:

$$T_{6_{d_{i_x}}} = -n_x p_y + n_y p_x$$

$$T_{6_{d_{i_y}}} = -o_x p_y + o_y p_x$$

$$T_{6_{d_{i_z}}} = -a_x p_y + a_y p_x$$

$$T_{6_{\partial_{i_x}}} = n_z$$

$$T_{6_{\partial_{i_y}}} = o_z$$

$$T_{6_{\partial_{i_z}}} = a_z$$

Es importante recordar que el jacobiano, solo puede ser usado de manera directa para relacionar las velocidades articulares (dq) con las velocidades cartesianas V. Para poder encontrar la relación en posición es necesario realizar una integración numérica:

$$V = J\dot{Q}$$
$$dT|_{\in \mathbb{R}^{6x1}} = JdQ$$
$$j^{-1}[T_{i-1} - T_i]|_{\in \mathbb{R}^{6x1}} = [Q_{i-1} - Q_i]$$
$$j^{-1}[T_{i-1} - T_i]|_{\in \mathbb{R}^{6x1}} - Q_i = Q_{i-1}$$

3. OBJETIVO DEL PROYECTO

Diseñar e implementar un controlador de seguimiento de trayectoria para el humanoide *RoboPhilo* que le permita imitar los movimientos específicos de las extremidades de una persona capturados por medio un sensor visual el sensor visual RGB-D Kinect.

Logramos realizar simulaciones de las extremidades del humanoide simultáneamente en el programa MATLAB, permitiéndonos verificar el correcto funciona mirto del análisis teórico previo, incluyendo la interacción entre la plataforma del sensor visual RGB-D Kinect y el programa MATLAB. Además el alcance de este proyecto fue que el humanoide *RoboPhilo* imitara los movimientos que se realizaban por el usuario mediante el sensor visual RGB-D Kinect.

4. DESARROLLO Y PROTOCOLO DE PRUEBAS

Este trabajo de grado propuso el uso de un humanoide comercial a bajo costo (ver Figura 4.1).

Figura 4.1 *RoboPhilo.* [9]

Durante el proceso de selección del humanoide encontramos varios kits de robot para armar, después de un análisis de varias plataformas enfocadas al desarrollo de humanoides, escogimos la plataforma robótica humanoide *RoboPhilo* diseñada por la compañía *RoboBrothers, inc*. *RoboPhilo* cumple con los requisitos necesarios en software y hardware, además de tener un precio razonable para ser adquirido. La plataforma escogida nos permite, con la ayuda de un sensor visual RGB-D Kinect y de programación en lenguaje *C,*

la imitación de los movimientos sencillos que realiza el ser humano. A continuación hablaremos de algunas de sus especificaciones:

- Altura: 13” (330.2mm).
- Peso: 1.2 kg (1200g). Incluye batería.
- 24 canales disponibles (20 utilizados para servomotores y 4 para sensores u otros actuadores).
- Servomotores SV 4032 (4.1 kg-cm), SV 4140 (6.5 kg-cm) y SV 2031(1.3 kg-cm).
- 8 kB de memoria flash.
- 20 grados de libertad.
- *Software Development Kit* (SDK), disponible para el desarrollo del programa en C para personalizar los comandos de control del robot autónomo.

El controlador principal cuenta con las siguientes características:

- 1 microcontrolador ATmega32 – 16 PU con las siguientes características:
  - 32KB de memoria flash.
  - 16MHz de frecuencia de operación.
  - Comunicación SPI, I2C, UART.
  - 8 canales de para conversión A/D a 10 bits de resolución.
  - 1 comparador analógico.
  - 2KB de memoria SRAM.
  - 1KB de memoria EEPROM.
  - Rango de voltaje de operación: 2.7V a 5.5V.
  - 3 Timers.
  - 4 canales PWM (pulse-width modulation).
- 3 canales para conversión Análogo/Digital.
- 3 Conectores de propósito general (acelerómetro, infrarrojo, giroscopio, etc.).
- 1 Conector para comunicación serial.

4.1. ARMADO DE ROBOT HUMANOIDE ROBOPHILO

El fabricante tiene dos versiones de *RoboPhilo*, *Ready-To- Walk Version* y *Kit Version*. Adquirimos *Kit Version*, dicha versión nos permite armar el humanoide siguiendo la guía de usuario, donde explican la secuencia de pasos que se deben seguir para completar el armado.

Figura 4.1.1 Armado del robot humanoide *RoboPhilo*

En la imagen de la derecha se puede observar cómo viene empacado el robot humanoide *RoboPhilo* y en la imagen de la izquierda luego después de armado.

Después de armar el humanoide completamente, se realizó el ajuste fino (fine tuning) de los servomotores mediante una interfaz desarrollada por el fabricante, con el fin de no generar movimientos que puedan exceder las capacidades del humanoide, de esta manera se comprueba el correcto funcionamiento del todos los servomotores.

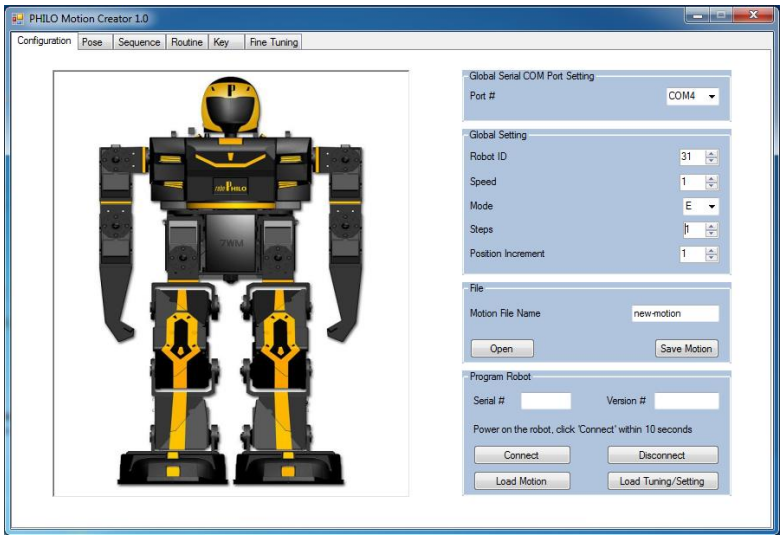


Figura 4.1.2 PHILO Motion Creator 1.0. [11]

Este programa de llama Philo Motion Creator 1.0, conectamos a *RoboPhilo* al computador usando el cable serial RS-232 y después en la última pestaña hacemos el ajuste fino.

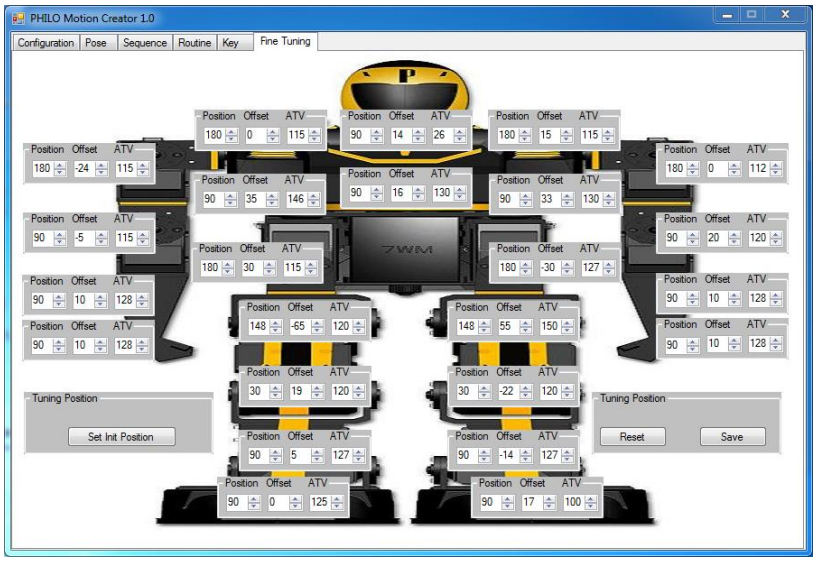


Figura 4.1.3 Ajuste fino. [11]

En el ajuste fino encontramos que podemos mover los 24 servos de cada articulación de *RoboPhilo* y cada articulación tiene tres parámetros, el primero es la posición, en él se puede cambiar la posición del servo, el segundo es el offset, con el definimos la posición de 0 grados o posición inicial de cada servo y el

tercero, es el ATV, con él se puede ajustar el ángulo de desplazamiento físico para el grado de posición de entrada.

4.2.MODELO CINEMÁTICO DE UN HUMANOIDE.

Después de verificar el correcto funcionamiento del humanoide procedemos a obtener un modelo matemático que nos permita describir analíticamente la secuencia que conlleva realizar un movimiento con el humanoide. Para ello obtuvimos los modelos: cinemático directo y cinemático inverso.

4.2.1. MODELO CINEMÁTICO DIRECTO

- Obtención de los parámetros D-H
  - Brazo Derecho

Articulación	θ (rad)	d (m)	a (m)	α (rad)
q_1	$q_1 = \frac{\pi}{2}$	0.02	0	$\frac{\pi}{2}$
q_2	$q_2 = 0$	0	0.055	π
q_3	$q_3 = \frac{\pi}{2}$	0	0.094	0

Tabla 4.2.1.1. Parámetros D-H brazo derecho.

  - Brazo Izquierdo

Articulación	θ (rad)	d (m)	a (m)	α (rad)
q_4	$q_4 = \frac{\pi}{2}$	0.02	0	$\frac{\pi}{2}$
q_5	$q_5 = 0$	0	0.055	π
q_6	$q_6 = \frac{\pi}{2}$	0	0.094	0

Tabla 4.2.1.2. Parámetros D-H brazo izquierdo.

  - Pierna izquierda

Articulación	θ (rad)	d (m)	a (m)	α (rad)
q_7	$q_7 = \frac{\pi}{2}$	0	0.04	$\frac{\pi}{2}$
q_8	$q_8 = \frac{\pi}{2}$	0	0.04	0
q_9	$q_9 = 0$	0	0.05	0

q_{10}	$q_{10} = \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
q_{11}	$q_{11} = \frac{\pi}{2}$	0	0.08	$\frac{\pi}{2}$

Tabla 4.2.1.3. Parámetros D-H pierna izquierda.

- Pierna derecha

Articulación	θ (rad)	d(m)	a (m)	α (rad)
q_{12}	$q_{12} = \frac{\pi}{2}$	0	0.04	$\frac{\pi}{2}$
q_{13}	$q_{13} = \frac{\pi}{2}$	0	0.04	0
q_{14}	$q_{14} = 0$	0	0.05	0
q_{15}	$q_{15} = \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
q_{16}	$q_{16} = \frac{\pi}{2}$	0	0.08	$\frac{\pi}{2}$

Tabla 4.2.1.4. Parámetros D-H pierna derecha.

Luego obtenemos las matrices de transformación homogénea para las articulaciones del humanoide. En nuestro caso los ángulos articulares son rotacionales y su matriz está representada por:

```
A = [cos(teta)   -cos(alfa)*sin(teta)     sin(alfa)*sin(teta)    a*cos(teta);
     sin(teta)    cos(alfa)*cos(teta)    -sin(alfa)*cos(teta)    a*sin(teta);
            0              sin(alfa)              cos(alfa)              d;
            0                     0                     0              1];
```

Figura 4.2.1.1. Matriz de transformación homogénea.

- Obtenemos la matriz de transformación homogénea para cada una de las articulaciones
  Dicha matriz se obtiene remplazando los parámetros D-H en la matriz de transformación homogénea correspondientes a cada articulación.

  - Matrices de transformación homogénea brazo derecho.
    Para simplificar la expresión, llamamos teta = t, alfa = al.

```
ABD01 =
[ cos(t1), -cos(al1)*sin(t1),  sin(al1)*sin(t1), a1*cos(t1)]
[ sin(t1),  cos(al1)*cos(t1), -sin(al1)*cos(t1), a1*sin(t1)]
[       0,          sin(al1),          cos(al1),         d1]
[       0,                 0,                 0,          1]
```

```
ABD12 =
[ cos(t2), -cos(al2)*sin(t2),  sin(al2)*sin(t2), a2*cos(t2)]
[ sin(t2),  cos(al2)*cos(t2), -sin(al2)*cos(t2), a2*sin(t2)]
[       0,          sin(al2),          cos(al2),         d2]
[       0,                 0,                 0,          1]
ABD23 =
[ cos(t3), -cos(al3)*sin(t3),  sin(al3)*sin(t3), a3*cos(t3)]
[ sin(t3),  cos(al3)*cos(t3), -sin(al3)*cos(t3), a3*sin(t3)]
[       0,          sin(al3),          cos(al3),         d3]
[       0,                 0,                 0,          1]
```

Figura 4.2.1.2. Matriz de transformación homogénea brazo derecho.

- Matrices de transformación homogénea brazo izquierdo.
  Para simplificar la expresión, llamamos teta = t y alfa = al

```
ABI04 =
[ cos(t4), -cos(al4)*sin(t4),  sin(al4)*sin(t4), a4*cos(t4)]
[ sin(t4),  cos(al4)*cos(t4), -sin(al4)*cos(t4), a4*sin(t4)]
[       0,          sin(al4),          cos(al4),         d4]
[       0,                 0,                 0,          1]
ABI45 =
[ cos(t5), -cos(al5)*sin(t5),  sin(al5)*sin(t5), a5*cos(t5)]
[ sin(t5),  cos(al5)*cos(t5), -sin(al5)*cos(t5), a5*sin(t5)]
[       0,          sin(al5),          cos(al5),         d5]
[       0,                 0,                 0,          1]
ABI56 =
[ cos(t6), -cos(al6)*sin(t6),  sin(al6)*sin(t6), a6*cos(t6)]
[ sin(t6),  cos(al6)*cos(t6), -sin(al6)*cos(t6), a6*sin(t6)]
[       0,          sin(al6),          cos(al6),         d6]
[       0,                 0,                 0,          1]
```

Figura 4.2.1.3. Matriz de transformación homogénea brazo izquierdo.

- Matrices de transformación homogénea pierna izquierda.
  Para simplificar la expresión, llamamos teta = t y alfa = al

```
API07 =
[ cos(t7), -cos(al7)*sin(t7),  sin(al7)*sin(t7), a7*cos(t7)]
[ sin(t7),  cos(al7)*cos(t7), -sin(al7)*cos(t7), a7*sin(t7)]
[       0,          sin(al7),          cos(al7),         d7]
[       0,                 0,                 0,          1]
API78 =
[ cos(t8), -cos(al8)*sin(t8),  sin(al8)*sin(t8), a8*cos(t8)]
[ sin(t8),  cos(al8)*cos(t8), -sin(al8)*cos(t8), a8*sin(t8)]
[       0,          sin(al8),          cos(al8),         d8]
[       0,                 0,                 0,          1]
API89 =
[ cos(t9), -cos(al9)*sin(t9),  sin(al9)*sin(t9), a9*cos(t9)]
[ sin(t9),  cos(al9)*cos(t9), -sin(al9)*cos(t9), a9*sin(t9)]
[       0,          sin(al9),          cos(al9),         d9]
[       0,                 0,                 0,          1]
```

```
API910 =
[ cos(t10), -cos(al10)*sin(t10),  sin(al10)*sin(t10), a10*cos(t10)]
[ sin(t10),  cos(al10)*cos(t10), -sin(al10)*cos(t10), a10*sin(t10)]
[        0,           sin(al10),           cos(al10),          d10]
[        0,                   0,                   0,            1]
API1011 =
[ cos(t11), -cos(al11)*sin(t11),  sin(al11)*sin(t11), a11*cos(t11)]
[ sin(t11),  cos(al11)*cos(t11), -sin(al11)*cos(t11), a11*sin(t11)]
[        0,           sin(al11),           cos(al11),          d11]
[        0,                   0,                   0,            1]
```

Figura 4.2.1.4. Matriz de transformación homogénea pierna izquierda.

- Matrices de transformación homogénea pierna derecha.
  Para simplificar la expresión, llamamos teta = t y alfa = al

```
APD012 =
[ cos(t12), -cos(al12)*sin(t12),  sin(al12)*sin(t12), a12*cos(t12)]
[ sin(t12),  cos(al12)*cos(t12), -sin(al12)*cos(t12), a12*sin(t12)]
[        0,           sin(al12),           cos(al12),          d12]
[        0,                   0,                   0,            1]
APD1213 =
[ cos(t13), -cos(al13)*sin(t13),  sin(al13)*sin(t13), a13*cos(t13)]
[ sin(t13),  cos(al13)*cos(t13), -sin(al13)*cos(t13), a13*sin(t13)]
[        0,           sin(al13),           cos(al13),          d13]
[        0,                   0,                   0,            1]
APD1314 =
[ cos(t14), -cos(al14)*sin(t14),  sin(al14)*sin(t14), a14*cos(t14)]
[ sin(t14),  cos(al14)*cos(t14), -sin(al14)*cos(t14), a14*sin(t14)]
[        0,           sin(al14),           cos(al14),          d14]
[        0,                   0,                   0,            1]
APD1415 =
[ cos(t15), -cos(al15)*sin(t15),  sin(al15)*sin(t15), a15*cos(t15)]
[ sin(t15),  cos(al15)*cos(t15), -sin(al15)*cos(t15), a15*sin(t15)]
[        0,           sin(al15),           cos(al15),          d15]
[        0,                   0,                   0,            1]
APD1516 =
[ cos(t16), -cos(al16)*sin(t16),  sin(al16)*sin(t16), a16*cos(t16)]
[ sin(t16),  cos(al16)*cos(t16), -sin(al16)*cos(t16), a16*sin(t16)]
[        0,           sin(al16),           cos(al16),          d16]
[        0,                   0,                   0,            1]
```

Figura 4.2.1.5. Matriz de transformación homogénea pierna derecha.

- Obtención del algoritmo de cinemática directa
  Se obtiene multiplicando de manera consecutiva las matrices de trasformación homogénea desde la base del robot hasta el efector final.

  - Algoritmo brazo derecho.
    `T03 = ABD01*ABD12*ABD23`

  - Algoritmo brazo izquierdo.
    `T06 = ABI04*ABI45*ABI56`

- Algoritmo pierna izquierda.
  `T011 = API07*API78*API89*API910*API1011`

- Algoritmo pierna derecha.
  `T106 = APD012*APD1213*APD1314*APD1415*APD1516`

4.3. ADQUISICION DE DATOS DEL SENSOR VISUAL RGB-D KINECT EN MATLAB.

Instalamos SDK para Windows 7 de tal manera que nos permitiera extraer información del *Kinect* por medio de MATLAB. Adquirimos una librería llamada *"Kinect For Windows"* de MATLAB para que el programa reconociera el sensor *Kinect* y que esta forma existiera compatibilidad entre ellos. (Ver anexo 1)

Luego procedimos a extraer datos relevantes para nuestro trabajo, como lo son: cámara de color, cámara de profundidad, dibujo de esqueleto, seguimiento de trayectoria y posicionamiento coordenado en el espacio.

Cámara de color

En la figura 4.3.1., podemos observar la habilitación de la cámara RGB (Red Green Blue) de 480x640 pixeles del sensor *Kinect.*

Figura 4.3.1 Camara de color del sensor visual RGB-D Kinect

Cámara de profundidad

En la figura 4.3.2., podemos observar por medio de tonos grises a que profundidad están los objetos en el espacio, identificándolos de la siguiente manera: los grises más oscuros representan un mayor acercamiento a la cámara y los grises más claros representan mayor alejamiento.

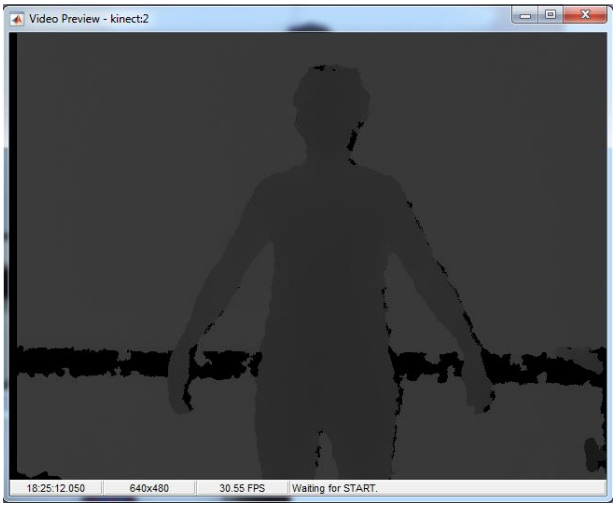
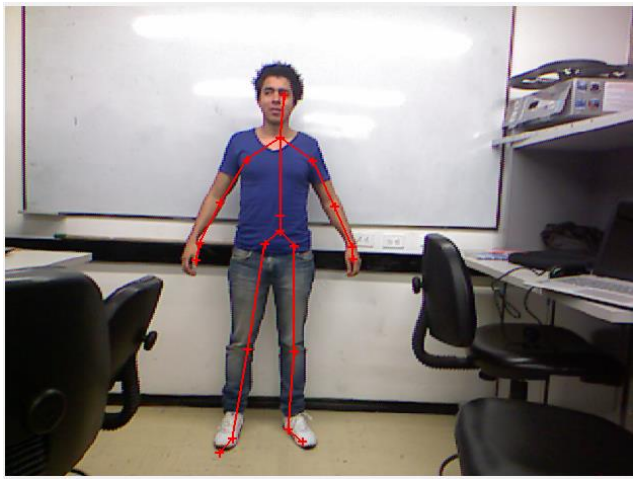


Figura 4.3.2 Camara de Profundidad del sensor visual RGB-D Kinect

Seguimiento de trayectoria

El dibujo esquelético de la persona se logra mediante el procesamiento de imagen realizado por el chip PrimeSense [7], el cual se encarga de comparar formas parecidas a las del ser humano como cabeza, torso, brazos y piernas, de esta manera identifica los puntos de la extremidades y los une para formar el esqueleto, luego de tener el esqueleto en este caso de color rojo por medio de las cámaras, Extraemos los puntos X, Y, Z de cada una de las articulaciones.



```
puntosxyz(:,:,1) =

   -0.3426   -0.2575    1.9536
   -0.3628   -0.2003    1.9926
   -0.3619    0.1189    1.9784
   -0.3171    0.3092    1.9126
   -0.5155    0.0310    1.9068
   -0.6661   -0.0215    1.7641
   -0.7615   -0.0295    1.5788
   -0.7870   -0.0297    1.4963
   -0.2036    0.0220    2.0498
   -0.1106   -0.1393    2.0921
    0.0539   -0.0538    1.9499
    0.1157   -0.0323    1.9081
   -0.4060   -0.3295    1.9183
   -0.4533   -0.6710    1.9667
   -0.4967   -0.9845    1.9984
   -0.4697   -1.0361    1.9325
   -0.2689   -0.3337    1.9759
   -0.2628   -0.6936    2.0340
   -0.3248   -0.9782    2.0993
   -0.3044   -1.0431    2.0436
```

Figura 4.3.3 Seguimiento de trayectoria y puntos coordenados.

4.4. PROCESAMIENTO DE DATOS ADQUIRIDOS DEL SENSOR VISUAL RGB-D KINECT EN MATLAB.

Luego de extraer los puntos coordenados de cada articulación debemos obtener una estimación del valor para cada articulación, este proceso puede tener múltiples o infinitas soluciones, por tal motivo se realizó de dos formas distintas obteniendo resultados diferentes. Dicho proceso se basa en partir de la posición del actuador final y con dichos valores cartesianos encontrar que valores para cada una de las articulaciones que posea la cadena articular, puedan satisfacer la posición del actuador final.

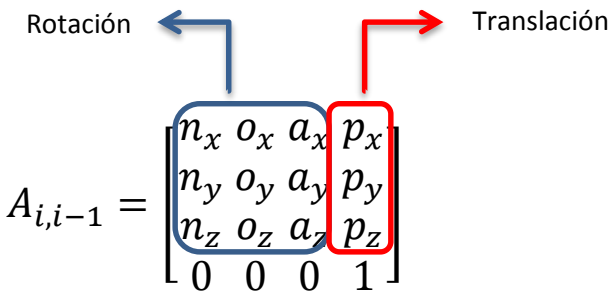
4.4.1.PROCESAMIENTO DE DATOS MEDIANTE CINEMÁTICA INVERSA

Una vez obtenida la matriz del jacobiano que describe la cadena cinemática y la posición del actuador final, es necesario obtener su inversa (pseudoinversa en caso de que la matriz no sea simétrica) para poder obtener los valores de las variables articulares. Los inconvenientes al realizar este procedimiento es que la cinemática inversa tiene múltiples soluciones o puede que matemáticamente nunca se llegue a una solución.

Posteriormente es necesario hallar los valores articulares para cada uno de los tiempos y al momento de realizar dicho procedimiento, puede llegar a no tener solución o tener múltiples soluciones y no se obtiene una solución coherente. La no convergencia de la ecuación con determinados valores de entrada demostró no ser eficiente para llevar a cabo este trabajo de grado, debido a esto, optamos por no utilizar el procesamiento de datos por cinemática inversa.

4.4.2.PROCESAMIENTO DE DATOS MEDIANTE TOOLBOX DE ROBÓTICA

El toolbox de robótica para MATLAB desarrollado por Peter Corke [11] permite realizar operaciones y procesos complejos por medio de funciones, obteniendo como resultado una solución finita. Se puede llegar a una solución partiendo de las matrices de transformación homogénea descritas anteriormente, dichas matrices poseen información relevante en cada una de sus columnas como lo son la rotación y la translación. En este proceso entraremos a modificar la translación de la matriz de transformación homogénea, donde se encuentra la posición final del actuador final.



Realizando el proceso hasta llegar al algoritmo de cinemática directa para poder obtener la relación del actuador final con respecto a toda la cadena articular, luego podemos encontrar las aproximaciones articulares para cada uno de los eslabones de la cadena cinemática inversa mediante la función “ikine” del toolbox para robótica. La garantía del correcto funcionamiento de la función “ikine” depende de la información recibida por dicha función. En nuestro caso la información necesaria para lograr una solución coherente es: tener una composición de la cadena cinemática con los respectivos parámetros D-H, posición del estado estacionario del humanoide al mismo tiempo los algoritmo de cinemática directa con la posición deseada del actuador final y la cantidad de articulaciones en las que se descompone el movimiento.

Una vez extraídos los puntos cartesianos de los actuadores finales es necesario remplazarlos en la sección de translación del algoritmo de cinemática directa, luego la matriz corregida es la indicada para poder dar

solución a los movimientos articulares mediante la función “ikine”. La función “ikine” da como resultado un valor para cada articulación de la cadena cinemática que en conjunto dan la posición del actuador final.

4.5. SIMULACIÓN DE HUMANOIDE EN MATLAB

Luego de haber terminado todo el proceso de adquisición y procesamiento de datos, nos enfocamos en emular los movimientos realizados por el usuario, para esto utilizamos la herramienta MATLAB y con ella hicimos la estructura de las extremidades del humanoide, partiendo de la estructura de cada extremidad por separado, para luego acoplarlas en un solo cuerpo.

Una de las complicaciones que encontramos al momento de realizar las simulaciones en MATLAB es que no se logró ejecutar dos códigos al mismo tiempo, es por eso que encontramos una solución y fue guardar los datos procesados, explicados anteriormente, para después reproducirlos en la simulación hecha en MATLAB de la estructura del humanoide.

4.5.1. ESTRUCTURA DE HUMANOIDE EN MATLAB

Para realizar la simulación de las extremidades de *RoboPhilo* en MATLAB utilizamos el toolbox de Peter Corke [11], dicho toolbox consiste en un conjunto de librerías para robótica que incluye herramientas de visión artificial para Matlab y con la ayuda del Toolbox de Peter Corke se realizó la simulación de las extremidades de *RoboPhilo* y verificamos que nuestras ecuaciones fueran las correctas.

La simulación consiste en estructurar cada una de las extremidades del humanoide a un eslabón inamovible, dicho eslabón funciona de base para cada una de las extremidades emulando el tronco del humanoide, posteriormente estructuramos cada una de las extremidades partiendo de los parámetros D-H extraídos anteriormente en el análisis teórico.

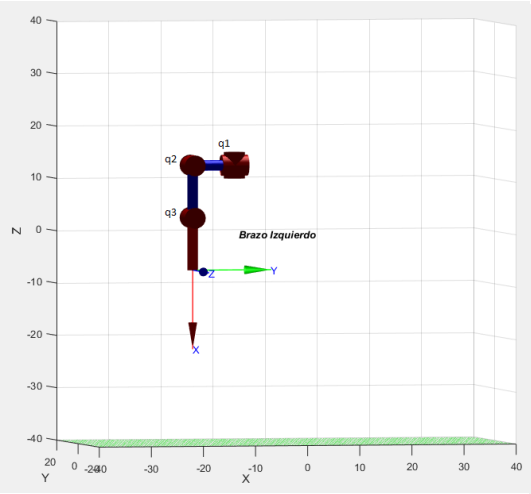


Figura 4.5.1.1 Brazo Izquierdo

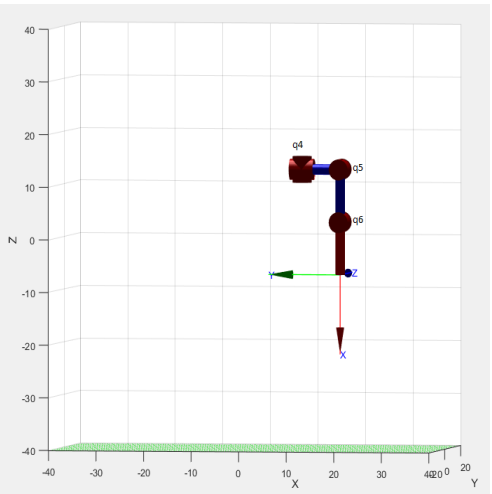


Figura 4.5.1.2 Brazo Derecho

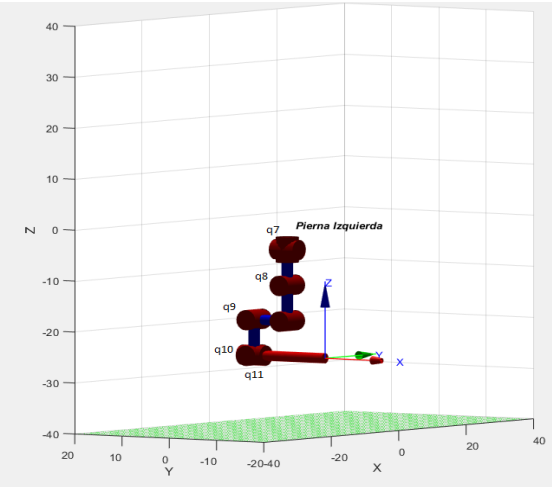


Figura 4. 5.1.3 Pierna Izquierda

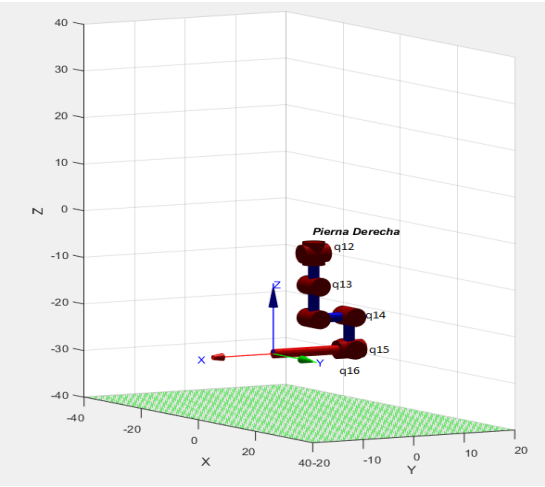


Figura 4.5.1.4 Pierna Derecha

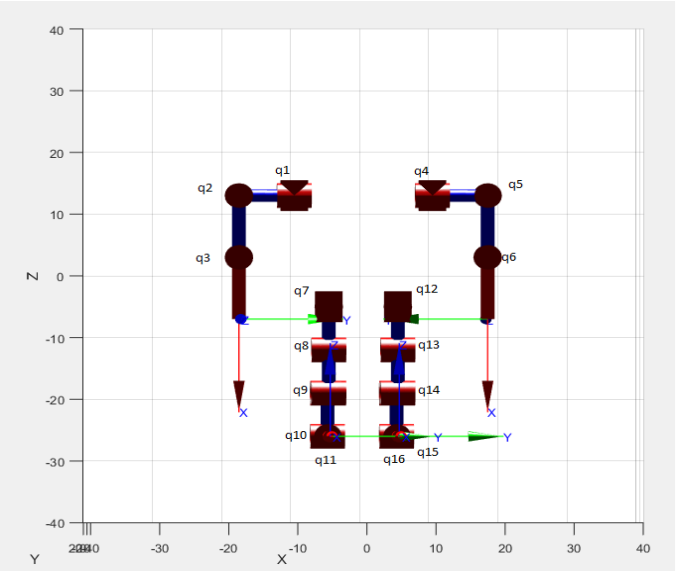


Figura 4.5.1.5 Cuerpo Completo Frontal

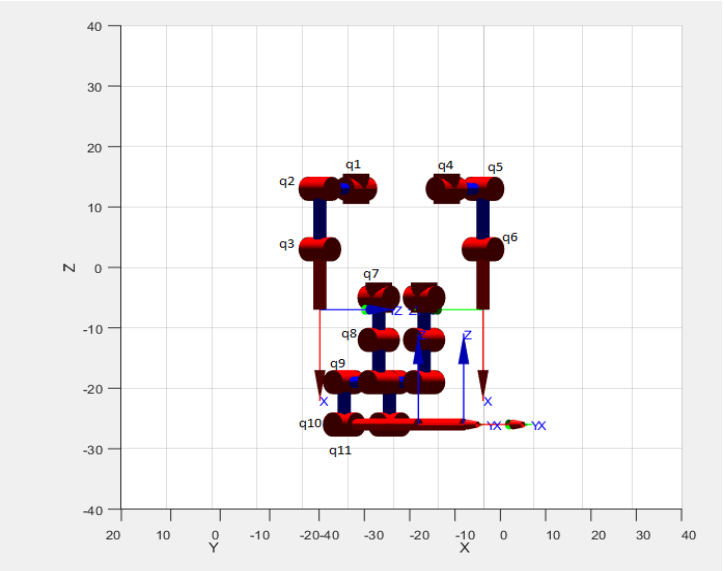


Figura 4.5.1.6 Cuerpo Completo Lateral

4.5.2. SIMULACIÓN DE MOVIMIENTOS DEL HUMANOIDE

Luego de hacer terminado todo el proceso de desarrollo del robot humanoide en MATLAB explicado anteriormente procedemos a obtener resultados de la simulación dando así un acercamiento al objetivo final del proyecto, partimos de los movimientos específicos descritos en nuestra propuesta de trabajo de grado y estos son los resultados obtenidos de la emulación de los movimientos.

- Estado estacionario
  Es la posición inicial para poder efectuar todos los movimientos descritos en nuestro proyecto.

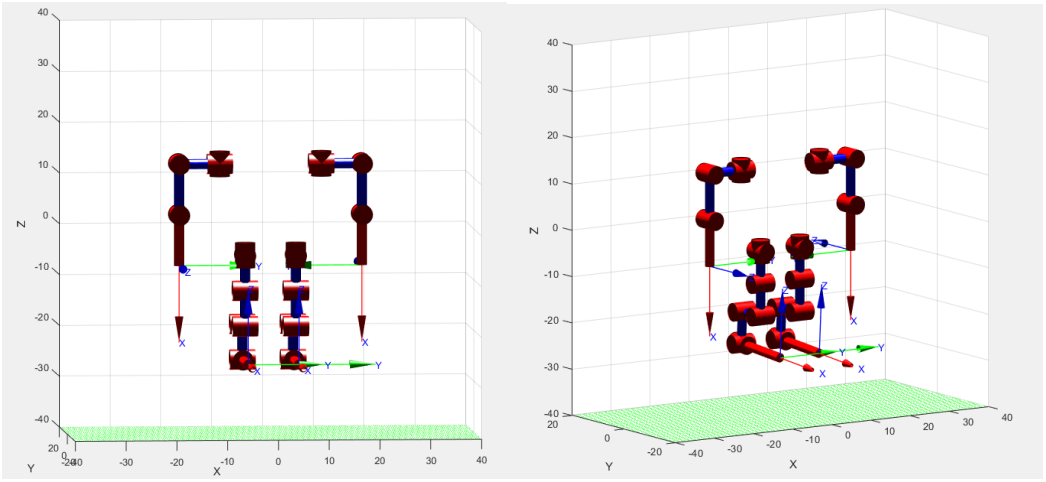


Figura 4.5.2.1 De pie: es el estado estacionario del humanoide.

- Flexion de rodillas

  En la imagen de la izquierda se puede observar el estado inicial del humanoide y en la imagen de la derecha se corre el programa y se puede observar la posicion final donde las rodillas se ven flexionadas hacia el frente.

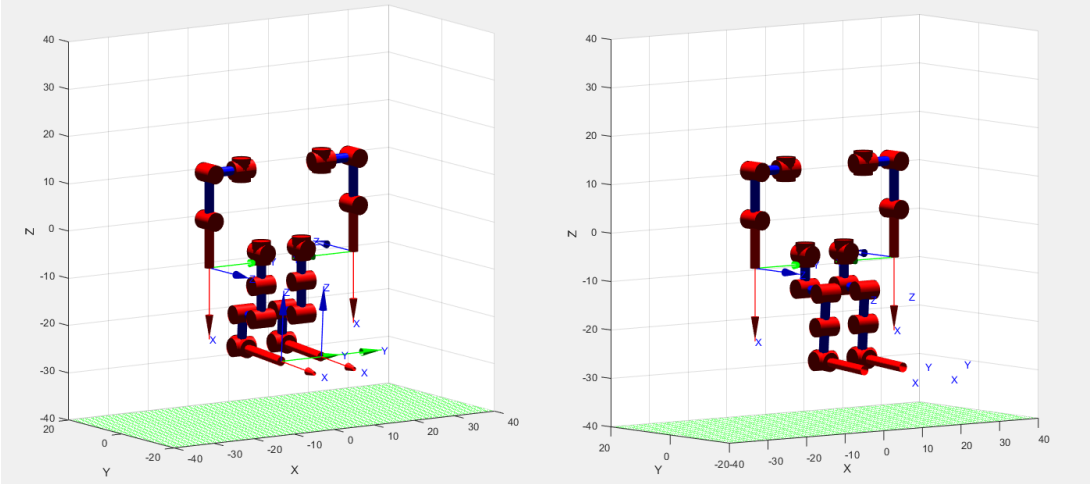


Figura 4.5.2.2 Flexion de rodillas.

- Brazos cruzados al pecho

  En la imagen de la izquierda se puede observar la posicion inicial del humanoide y en la imagen de la derecha se corre el programa y se puede observar los brazon flexionados hacia adelante al frente del pecho.

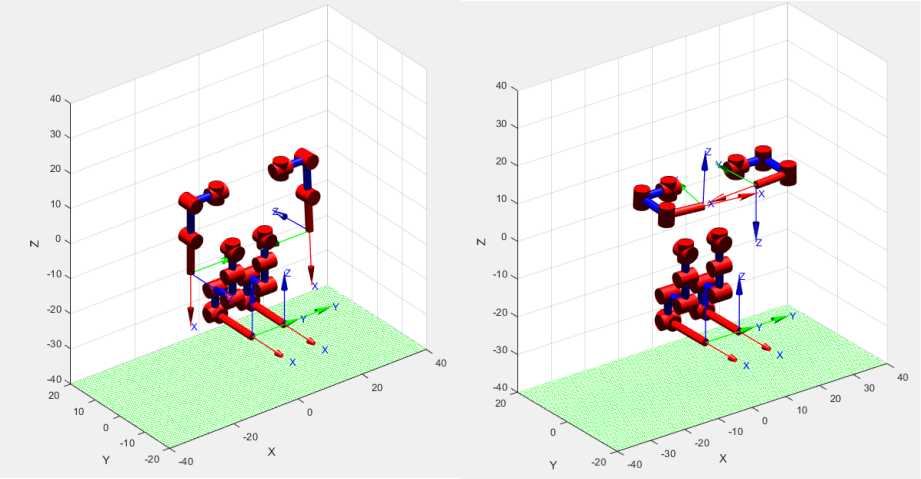


Figura 4.5.2.3 Brazos cruzados al pecho.

- Brazos completamente estirados hacia arriba
  En la imagen de la izquierda se puede observar la posición inicial del humanoide y en la imagen de la derecha corremos el programa y se puede observar los brazos completamente extendidos hacia arriba.

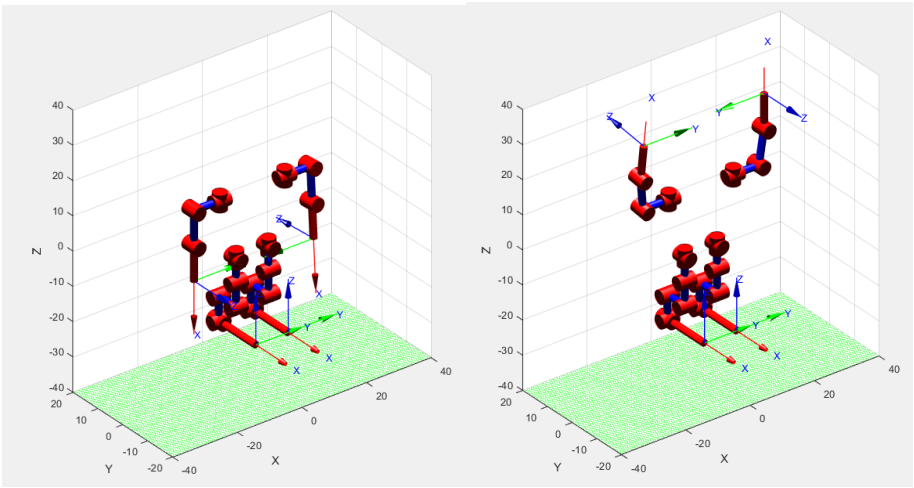


Figura 4.5.2.4 Brazos completamente estirados hacia arriba.

- Brazos completamente estirados hacia los lados
  En la imagen de la izquierda se puede observar la posición inicial del humanoide y en la imagen de la derecha corremos el programa y se puede observar los brazos completamente extendidos hacia los lados.

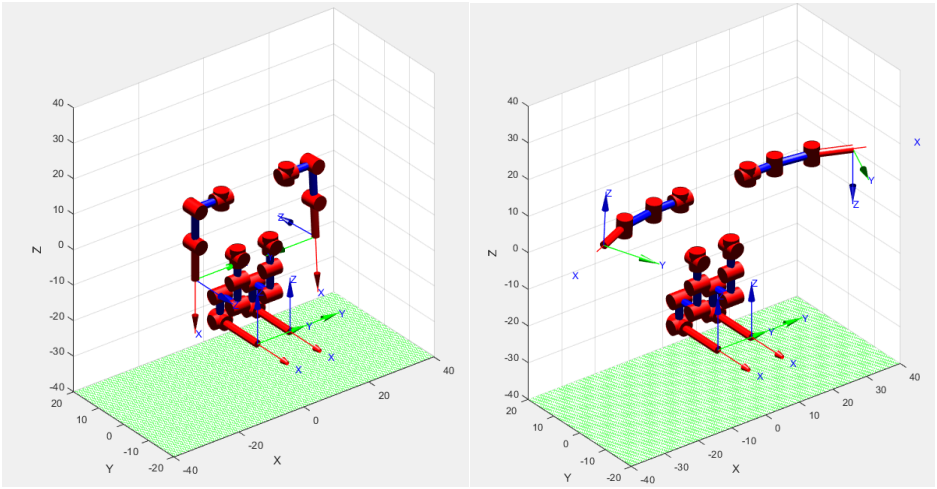


Figura 4.5.2.5 Brazos completamente estirados hacia los lados.

- Cuclillas

  En la imagen de la izquierda se puede observar la posición inicial del humanoide y en la imagen de la derecha corremos el programa y se puede observar las piernas del humanoide flexionadas donde las rodillas llegan lo más cercano al pecho.

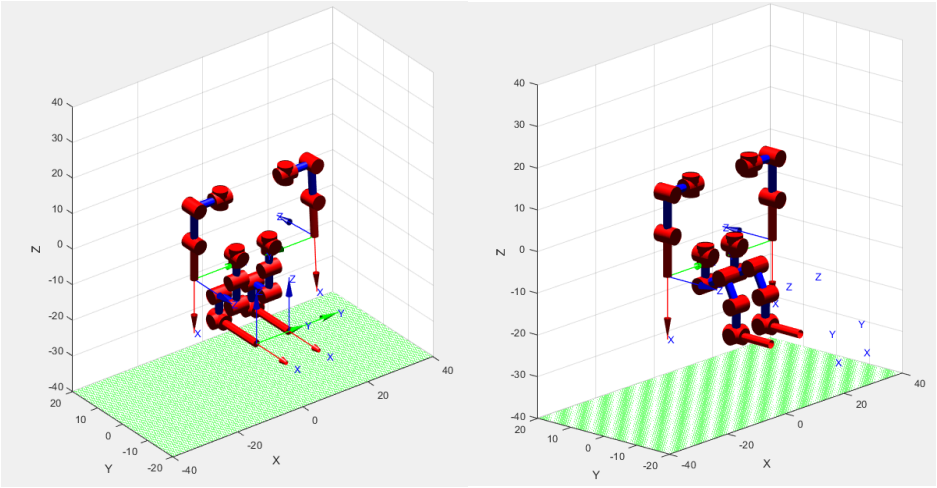


Figura 4.5.2.6 Cuclillas.

4.6. INTERACCION DEL HUMANOIDE CON EL SENSOR VISUAL RGB-D KINECT

4.6.1. CONFIGURACIÓN DE ENTORNO DE WINDOWS Y VERIFICACION DEL SENSOR VISUAL RGB-D KINECT.

Luego de terminar la simulación del humanoide en MATLAB, y verificar que funcionara correctamente, investigamos sobre la interacción del humanoide con el sensor visual RGB-D Kinect y cómo lograr realizar los movimientos, para esto descargamos e instalamos Visual C# 2010 Express que es un programa de desarrollo para sistemas operativos Windows, desarrollado y distribuido por Microsoft Corporation, Es de carácter gratuito fácil de manejar y soporta lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, ideal para nuestra aplicación con el robot humanoide.

Una vez instalado Visual C# 2010 Express, utilizamos el SDK de Kinect versión v1.7 previamente instalado, para después poder interactuar con el programa Skeleton Basics-WPF; este programa realiza un seguimiento de trayectoria que nos permite observar las articulaciones además de poder extraer la posición coordenada X, Y, Z, en metros de cada una de las articulaciones nos muestra el esqueleto y el seguimiento de trayectoria de nuestro cuerpo una vez conectado el sensorvisual RGB-D Kinect al computador.

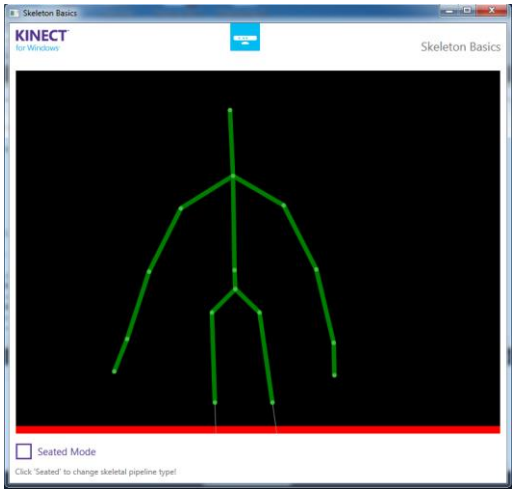


Figura 4.6.1.1 Tracking esquelético utilizando WPF.

Podemos observar una ventana de color negro con una silueta verde, esa silueta representa el esqueleto cuando estamos al frente del sensor visual RGB-D Kinect, después ajustamos el ángulo de la cámara del sensor visual RGB-D Kinect para ver en la ventana todo nuestro cuerpo desde la cabeza hasta los pies y verificamos su correcto funcionamiento moviendo cada articulación viendo reflejado los mismos movimientos en el programa.

4.6.2. INSTALACIÓN DEL SDK DE *ROBOPHILO* Y CONFIGURACIÓN DE MODULOS DEL SENSOR VISUAL RGB-D KINECT.

Luego de verificar el correcto funcionamiento del sensor visual RGB-D Kinect con el programa Visual C# 2010 Express nos enfocamos en hacer uso correcto del SDK de *RoboPhilo*, este programa fue suministrado por la compañía *RoboBrothers, inc.* Y es el encargado de permitir la comunicación entre el computador y el robot, asignándole variables a cada articulación, seguimos las instrucciones del SDK de *RoboPhilo*.

Después de ver el correcto funcionamiento del SDK de *RoboPhilo* empezamos a trabajar con Visual C# 2010 express y compilamos el programa descrito anteriormente, verificamos su funcionamiento y que estuviera conectado al puerto (COM) deseado.

4.6.3. INTERACCION DEL SENSOR VISUAL RGB-D KINECT CON *ROBOPHILO*

La interacción entre la plataforma del humanoide y el sensor RGB-D Kinect no fue posible por medio de MATLAB, debido a que el software interno del humanoide no es compatible con MATLAB, para poder realizar una interacción eficiente entre el humanoide y el sensor RGB-D Kinect fue necesario utilizar librerías de código libre proporcionadas por Microsoft. Partiendo de código necesario para acceder a las

utilidades que ofrece el sensor y poder complementarlas con el código necesario para la extracción de datos y su correcto procesamiento.

Conectamos el Kinect al computador y prendimos a *RoboPhilo,* compilamos el programa en visual C# 2010 express y observamos que funcionara correctamente, para inicializar el robot ponemos nuestra mano derecha arriba del hombro y de esta manera el robot humanoide empieza a imitar nuestros movimientos. Para parar el sistema nos agachamos un poco y de esta manera el programa sabe que debe dejar de ejecutar el programa, no se pueden realizar movimientos que incluyan las piernas donde el usuario se agache más de 20 cm, debido a que, el SDK del humanoide no permite que el humanoide realice movimientos que estén fuera de su capacidad o puedan presentar un peligro para la plataforma.

5. ANÁLISIS DE RESULTADOS

5.1. RESULTADOS DE LA SIMULACIÓN.

Los parámetros más relevantes y de los cuales se podía llevar un análisis que demostrara el cumplimiento del objetivo de nuestro proyecto son: los errores cartesianos y articulares, la variación de las articulaciones en el tiempo y rango de operación. En simulación el tiempo de respuesta no será una variable a tener en cuenta, como se explicó anteriormente al compilar dos códigos, la extracción de puntos coordenados y las gráficas de las articulaciones, en MATLAB simultáneamente no fue posible, por ende la solución más favorable para permitir la simulación fue:

Primero, extraer los datos de la posición cartesiana de las articulaciones, luego mediante la cinemática directa obtener el algoritmo de transformación homogénea y en dicho algoritmo se remplazaba la posición del actuador final, ya con los parámetros completos que necesita la función *ikine* del toolbox de robótica procedíamos a obtener los posibles valores de las articulaciones que solucionaran el problema cinemático inverso. Segundo, una vez obtenidos los valores de las articulaciones con la posible solución verificábamos que no excedieran los límites para cada articulación del humanoide y procedíamos a realizar el movimiento en simulación observando el comportamiento de toda la secuencia.

Se dedujo que el tiempo de respuesta en simulación no es relevante debido a, dicho tiempo sería el tiempo que se demora MATLAB en leer los datos de una matriz ya establecida, es decir, no sería un parámetro propio de la simulación sino del programa como tal, por ende el tiempo de respuesta en simulación no será tenido en cuenta.

5.1.1. ERROR CARTESIANO EN ESTADO ESTACIONARIO.

Para medir el error de las variables cartesianas X, Y, Z, comparamos los puntos en el espacio que suministra el sensor visual RGB-D Kinect a MATLAB dadas en metros del actuador final y se planteaba una similitud a escala con la simulación del humanoide, los puntos extraídos del sensor visual RGB-D Kinect no pueden ser comparados directamente con la simulación, debido a que la representación gráfica se realizó a una escala proporcional del humanoide, entonces las distancias en valor numérico no son comparables.

El error en estado estacionario se logró comparando la posición final del humanoide con la posición final del usuario, de esta manera sabíamos que posición debía tomar determinada extremidad y obtener el error

resultante, se realizó el procedimiento para cada uno de los movimientos para finalmente obtener un error promedio. Los valores obtenidos están referenciados de la siguiente manera:

```
puntosxyz(:,:,1) =

  -0.3426   -0.2575    1.9536 -> q1
  -0.3628   -0.2003    1.9926 -> q2
  -0.3619    0.1189    1.9784 -> q3
  -0.3171    0.3092    1.9126 -> q4
  -0.5155    0.0310    1.9068 -> q5
  -0.6661   -0.0215    1.7641 -> q6
  -0.7615   -0.0295    1.5788 -> q7
  -0.7870   -0.0297    1.4963 -> q8
  -0.2036    0.0220    2.0498 -> q9
  -0.1106   -0.1393    2.0921 -> q10
   0.0539   -0.0538    1.9499 -> q11
   0.1157   -0.0323    1.9081 -> q12
  -0.4060   -0.3295    1.9183 -> q13
  -0.4533   -0.6710    1.9667 -> q14
  -0.4967   -0.9845    1.9984 -> q15
  -0.4697   -1.0361    1.9325 -> q16
  -0.2689   -0.3337    1.9759 N/A
  -0.2628   -0.6936    2.0340 N/A
  -0.3248   -0.9782    2.0993 N/A
  -0.3044   -1.0431    2.0436 N/A
```

Figura 5.1.1.1. Valores coordenados de la posición final para cada articulación.

Donde la primera columna de izquierda a derecha corresponde a la posición en X de cada una de las articulaciones, de igual manera la segunda corresponde a la posición en Y, la tercera a la posición en Z. por otro lado q1, q2 y q3 corresponden al brazo derecho; q4, q5 y q6 corresponden al brazo izquierdo; q7, q8, q9 q10 y q11 corresponden a la pierna izquierda y por ultimo q12, q13, q14, q15 y q16 corresponden a la pierna derecha, las ultimas variables no aplican porque corresponden a la cabeza y al torso y en este trabajo de grado no serán tenidas en cuenta.

- Flexión de rodillas.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.1.2. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 2%.

```
puntosxyz =

    0.0847    0.1248    3.0451
    0.0863    0.1863    3.0287
    0.0951    0.4938    2.9647
    0.1143    0.6801    2.8971
   -0.0725    0.4095    2.9513
   -0.1576    0.1815    2.9603
   -0.1998   -0.0260    2.9046
   -0.2034   -0.1034    2.8913
    0.2515    0.3916    2.9829
    0.3221    0.1437    3.0360
    0.3562   -0.0704    2.9646
    0.3601   -0.1471    2.9480
    0.0092    0.0531    3.0516
   -0.1025   -0.2922    2.8913
   -0.0960   -0.6472    3.0561
   -0.1015   -0.7224    3.0436
    0.1565    0.0523    3.0660
    0.2313   -0.2953    2.9307
    0.2028   -0.5958    3.0418
    0.2086   -0.6424    2.9675
```

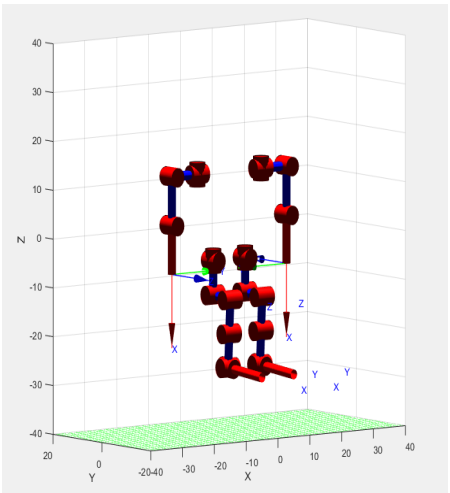
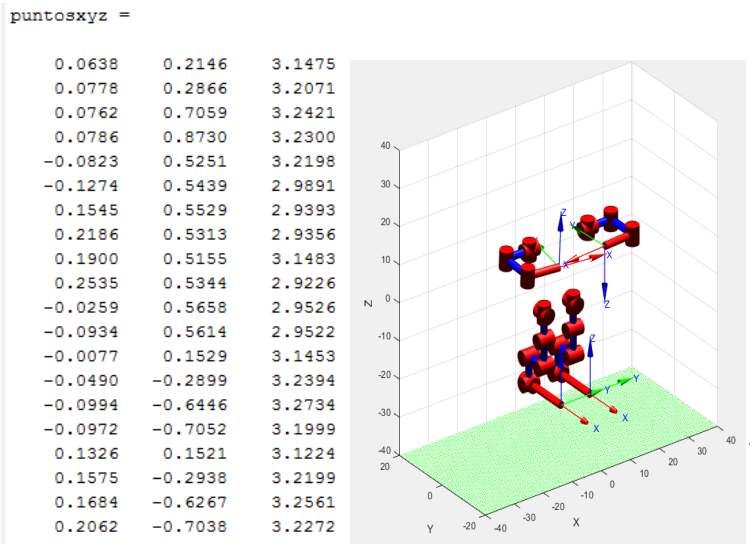


Figura 5.1.1.2. Error cartesiano en simulación flexión de rodillas.

- Brazos cruzados al pecho.

Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.1.3. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 2%.

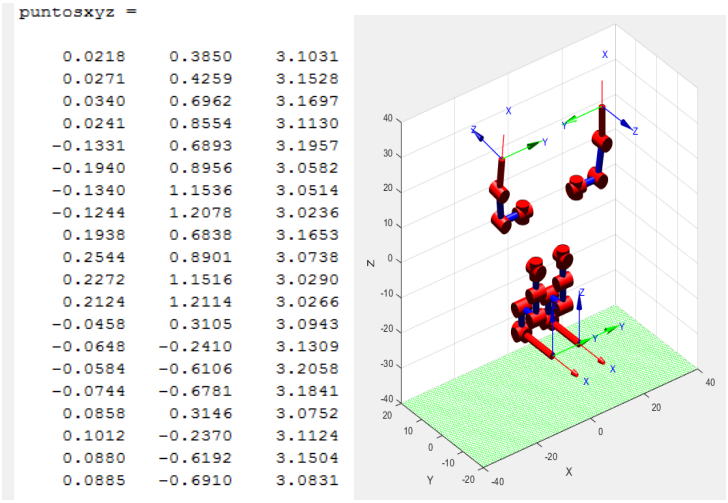


```
puntosxyz =

    0.0638    0.2146    3.1475
    0.0778    0.2866    3.2071
    0.0762    0.7059    3.2421
    0.0786    0.8730    3.2300
   -0.0823    0.5251    3.2198
   -0.1274    0.5439    2.9891
    0.1545    0.5529    2.9393
    0.2186    0.5313    2.9356
    0.1900    0.5155    3.1483
    0.2535    0.5344    2.9226
   -0.0259    0.5658    2.9526
   -0.0934    0.5614    2.9522
   -0.0077    0.1529    3.1453
   -0.0490   -0.2899    3.2394
   -0.0994   -0.6446    3.2734
   -0.0972   -0.7052    3.1999
    0.1326    0.1521    3.1224
    0.1575   -0.2938    3.2199
    0.1684   -0.6267    3.2561
    0.2062   -0.7038    3.2272
```

Figura 5.1.1.3. Error cartesiano en simulación brazos cruzados al pecho.

- Brazos completamente estirados hacia arriba.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.1.4. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 2%.



```
puntosxyz =

    0.0218    0.3850    3.1031
    0.0271    0.4259    3.1528
    0.0340    0.6962    3.1697
    0.0241    0.8554    3.1130
   -0.1331    0.6893    3.1957
   -0.1940    0.8956    3.0582
   -0.1340    1.1536    3.0514
   -0.1244    1.2078    3.0236
    0.1938    0.6838    3.1653
    0.2544    0.8901    3.0738
    0.2272    1.1516    3.0290
    0.2124    1.2114    3.0266
   -0.0458    0.3105    3.0943
   -0.0648   -0.2410    3.1309
   -0.0584   -0.6106    3.2058
   -0.0744   -0.6781    3.1841
    0.0858    0.3146    3.0752
    0.1012   -0.2370    3.1124
    0.0880   -0.6192    3.1504
    0.0885   -0.6910    3.0831
```

Figura 5.1.1.4. Error cartesiano en simulación brazos completamente estirados hacia arriba.

- Brazos completamente estirados hacia los lados.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.1.5. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 2%.

```
puntosxyz =

    0.0605    0.2955    2.9945
    0.0634    0.3531    3.0452
    0.0604    0.6834    3.0447
    0.0552    0.8595    3.0153
   -0.1156    0.6101    3.0700
   -0.3371    0.6560    3.0309
   -0.5521    0.6834    2.9407
   -0.6133    0.6903    2.9354
    0.2437    0.5949    3.0448
    0.4818    0.6270    3.0242
    0.6750    0.6448    2.9861
    0.7354    0.6443    2.9733
   -0.0160    0.2157    2.9888
   -0.0049   -0.2847    3.0527
    0.0112   -0.6480    3.0736
    0.0292   -0.7090    3.0368
    0.1366    0.2146    2.9740
    0.1241   -0.2678    3.0431
    0.0940   -0.6402    3.0772
    0.0776   -0.7056    3.0480
```

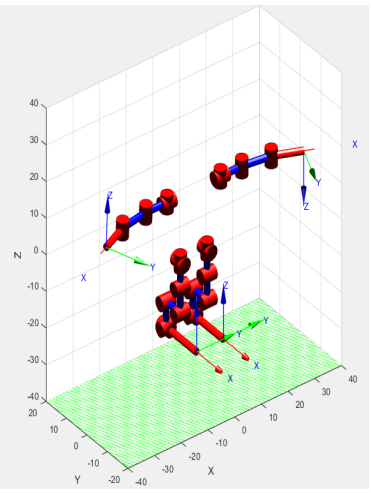


Figura 5.1.1.5. Error cartesiano en simulación brazos completamente estirados hacia los lados.

- Cuclillas.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.1.6. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 2%.

```
puntosxyz =

    0.0769   -0.2692    2.9405
    0.0784   -0.2129    2.9267
    0.0872    0.0690    2.8707
    0.0974    0.2681    2.8233
   -0.0826   -0.0153    2.8405
   -0.1567   -0.2568    2.8500
   -0.1688   -0.3999    2.6069
   -0.1909   -0.4578    2.6106
    0.2459   -0.0375    2.8978
    0.2969   -0.2815    2.8964
    0.2354   -0.4161    2.6560
    0.2465   -0.4603    2.6007
   -0.0055   -0.3387    2.9471
   -0.1402   -0.4308    2.6265
   -0.1338   -0.6348    2.8602
   -0.1224   -0.6813    2.7861
    0.1554   -0.3462    2.9589
    0.2539   -0.4647    2.6336
    0.2029   -0.6496    2.9235
    0.1866   -0.7283    2.9426
```

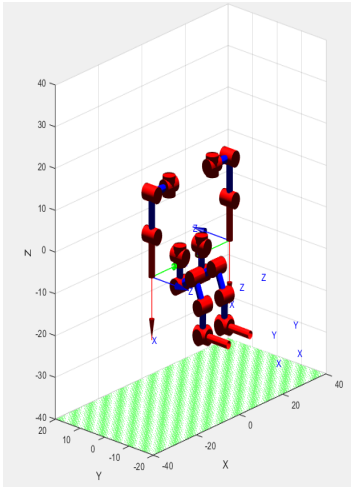


Figura 5.1.1.6. Error cartesiano en simulación cuclillas.

5.1.2.ERROR ARTICULAR EN ESTADO ESTACIONARIO.

Se realizó una estimación del error articular analizando por separado cada una de las articulaciones que en conjunto ejecutaban determinado movimiento, determinando que el error en la posición articular seria mayor que el error cartesiano debido a la cantidad de puntos observados para calcular el error. Para encontrar el error articular en cada uno de los movimientos, comparamos el valor de cada una de las articulaciones mostradas con el sensor visual RGB-D Kinect con el valor que obtuvimos de la simulación en MATLAB. Los valores correspondientes a cada articulación serán analizados de la siguiente manera:

```
Qs_finales =

   -1.5708 -> q1
    0.1659 -> q2
    0.1659 -> q3
    1.4451 -> q4
    0.1659 -> q5
    0.1659 -> q6
    1.5708 -> q7
    2.0378 -> q8
   -1.5708 -> q9
   -0.4712 -> q10
         0 -> q11
   -1.5708 -> q12
   -1.5708 -> q13
    2.0378 -> q14
   -1.5708 -> q15
   -0.4712 -> q16
```

Figura 5.1.2.1. Valores de cada articulación en su posición final

Donde q1, q2 y q3 corresponden al brazo derecho; q4, q5 y q6 corresponden al brazo izquierdo; q7, q8, q9 q10 y q11 corresponden a la pierna izquierda y por ultimo q12, q13, q14, q15 y q16 corresponden a la pierna derecha.

- Flexión de rodillas.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.2.2. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 3%.

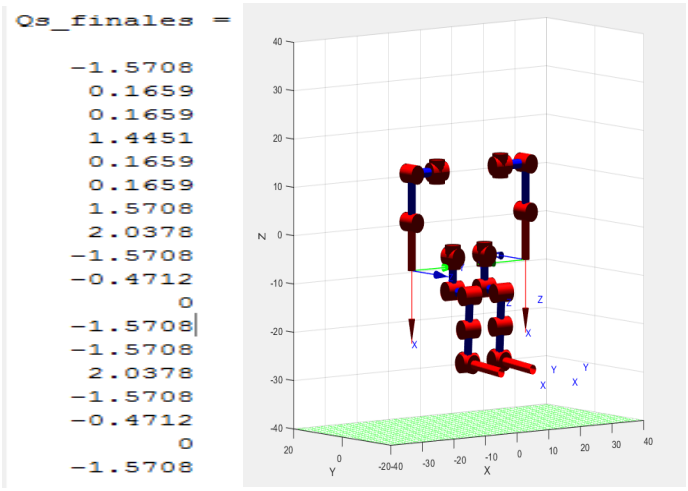


Figura 5.1.2.2. Error articular en simulación en simulación flexión de rodillas.

- Brazos cruzados al pecho.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.2.3. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de %.

```
qs_finales =

         0
         0
    1.5080
   -0.0628
         0
    1.4500
    1.5708
         0
   -1.5708
    1.5708
         0
   -1.5708
   -1.5708
         0
   -1.5708
    1.5708
         0
   -1.5708
```

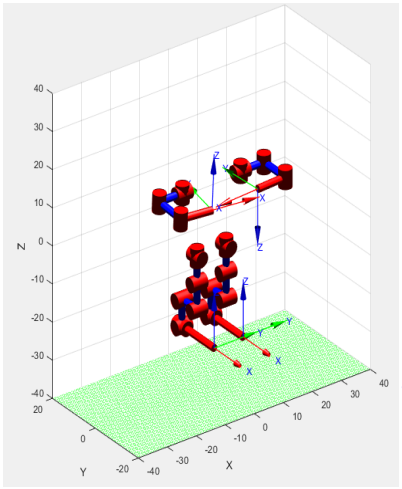


Figura 5.1.2.3. Error articular en simulación brazos cruzados al pecho.

- Brazos completamente estirados hacia arriba.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.2.4. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 3%.

```
qs_finales =

   -1.5708
    0.1659
    0.1659
    1.4451
    0.1659
    0.1659
    1.5708
         0
   -1.5708
    1.5708
         0
   -1.5708
   -1.5708
         0
   -1.5708
    1.5708
         0
   -1.5708
```

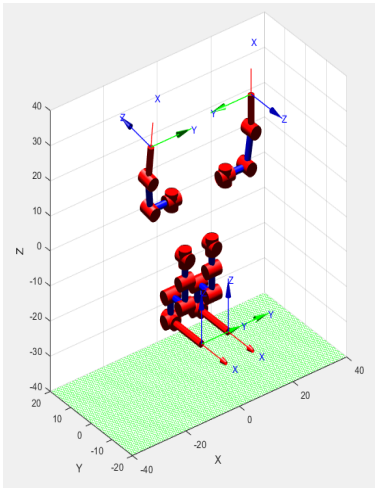


Figura 5.1.2.4. Error articular en simulación brazos completamente estirados hacia arriba.

- Brazos completamente estirados hacia los lados.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.1.6. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 3%.

```
qs_finales =

         0
    1.4175
    0.2511
   -0.0628
    1.4175
    0.2511
    1.5708
         0
   -1.5708
    1.5708
         0
   -1.5708
   -1.5708
         0
   -1.5708
    1.5708
         0
   -1.5708
```

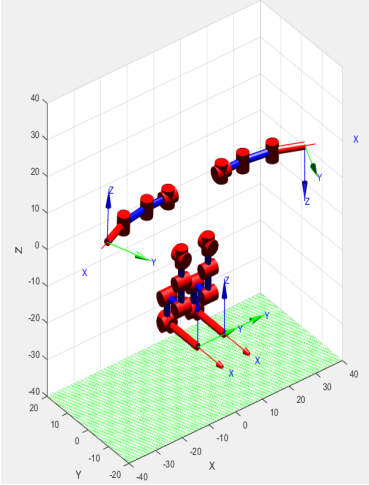


Figura 5.1.2.5. Error articular en simulación brazos completamente estirados hacia los lados.

- Cuclillas.
  Se analizó gráficamente la posición final del actuador final para estimar un error de posición final, en la figura 5.1.1.6. Podemos observar que la posición del usuario y la posición del robot humanoide simulado tienen un error aproximado de 8%.

```
qs_finales =

    1.5708
         0
         0
   -1.5708
         0
         0
    1.5708
    2.0378
   -1.5708
   -0.4712
         0
   -1.5708
   -1.5708
    2.0378
   -1.5708
   -0.4712
         0
   -1.5708
```

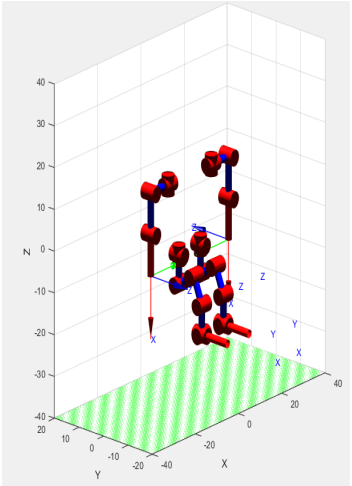


Figura 5.1.2.6. Error articular en simulación cuclillas.

5.1.3. VARIACIÓN DE LAS ARTICULACIONES EN EL TIEMPO.

La simulación nos permite obtener la variación de las articulaciones para cada movimiento con respecto al tiempo, debido a la forma de procesar los datos en simulación se obtiene en la mayoría de los casos una relación lineal y al no tener en cuenta el tiempo de respuesta cada articulación tiene un valor diferente para cada tiempo.

A continuación se observara la variación de las articulaciones que tiene más relevancia en cada movimiento:

- Flexión de rodillas.
  Podemos observar que solo dos articulaciones (q8 y q9) tienen un cambio significativo, en su valor articular con respecto al tiempo, debido a que son las articulaciones principalmente necesarias para lograr dicho movimiento.

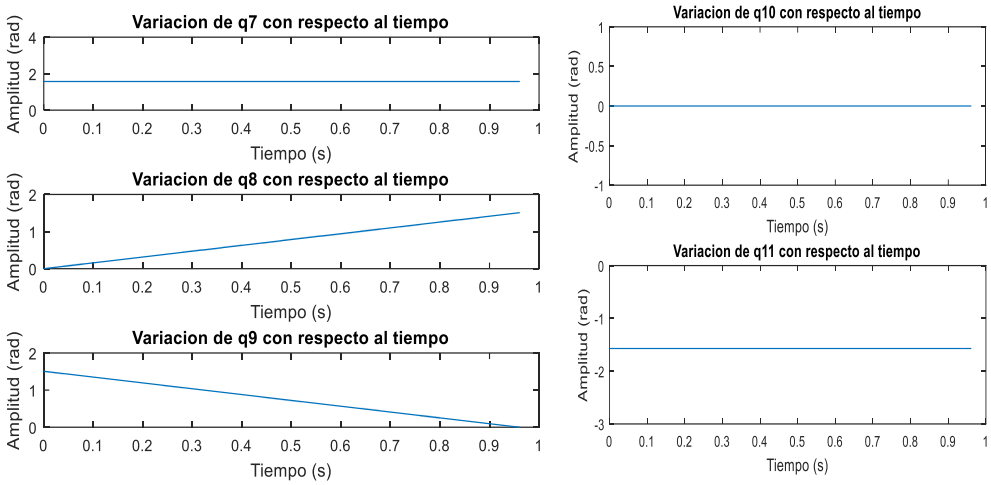


Figura 5.1.3.1. Variación de las articulaciones de la pierna izquierda.

- Brazos cruzados al pecho.
  Podemos observar que solo dos articulaciones en cada brazo (brazo derecho: q1 y q3, brazo izquierdo q4 y q6) tienen un cambio significativo, en su valor articular con respecto al tiempo, debido a que son las articulaciones principalmente necesarias para lograr dicho movimiento.

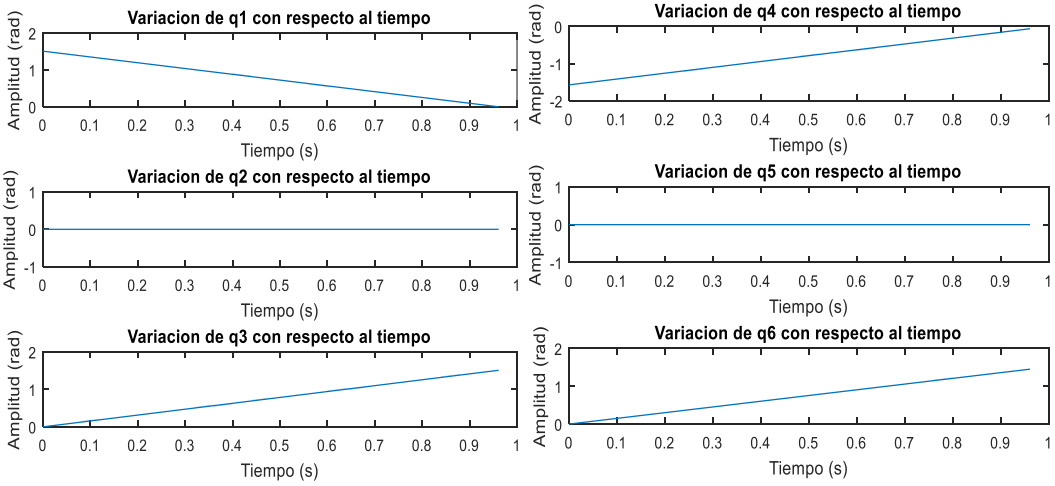


Figura 5.1.3.2. Variación de las articulaciones de los brazos al pecho.

- Brazos completamente estirados hacia arriba.
  Podemos observar todas las articulaciones tienen un cambio significativo en su valor articular con respecto al tiempo, debido a que todas las articulaciones son necesarias para lograr dicho movimiento.

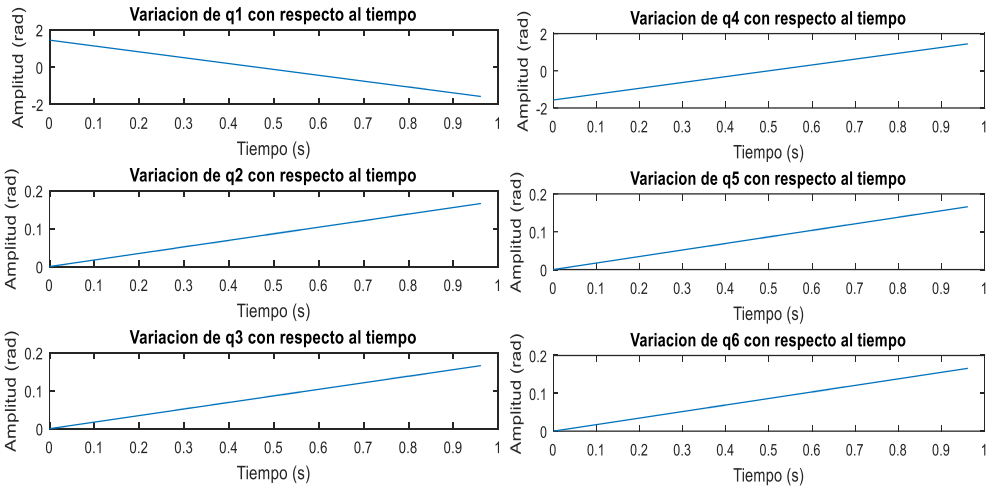


Figura 5.1.3.3. Variación de las articulaciones de los brazos estirados hacia arriba.

- Cuclillas.

  Podemos observar que solo dos articulaciones en cada pierna (pierna izquierda: q8 y q9, pierna derecha: q13 y q14) tienen un cambio significativo, en su valor articular con respecto al tiempo, debido a que son las articulaciones principalmente necesarias para lograr dicho movimiento.

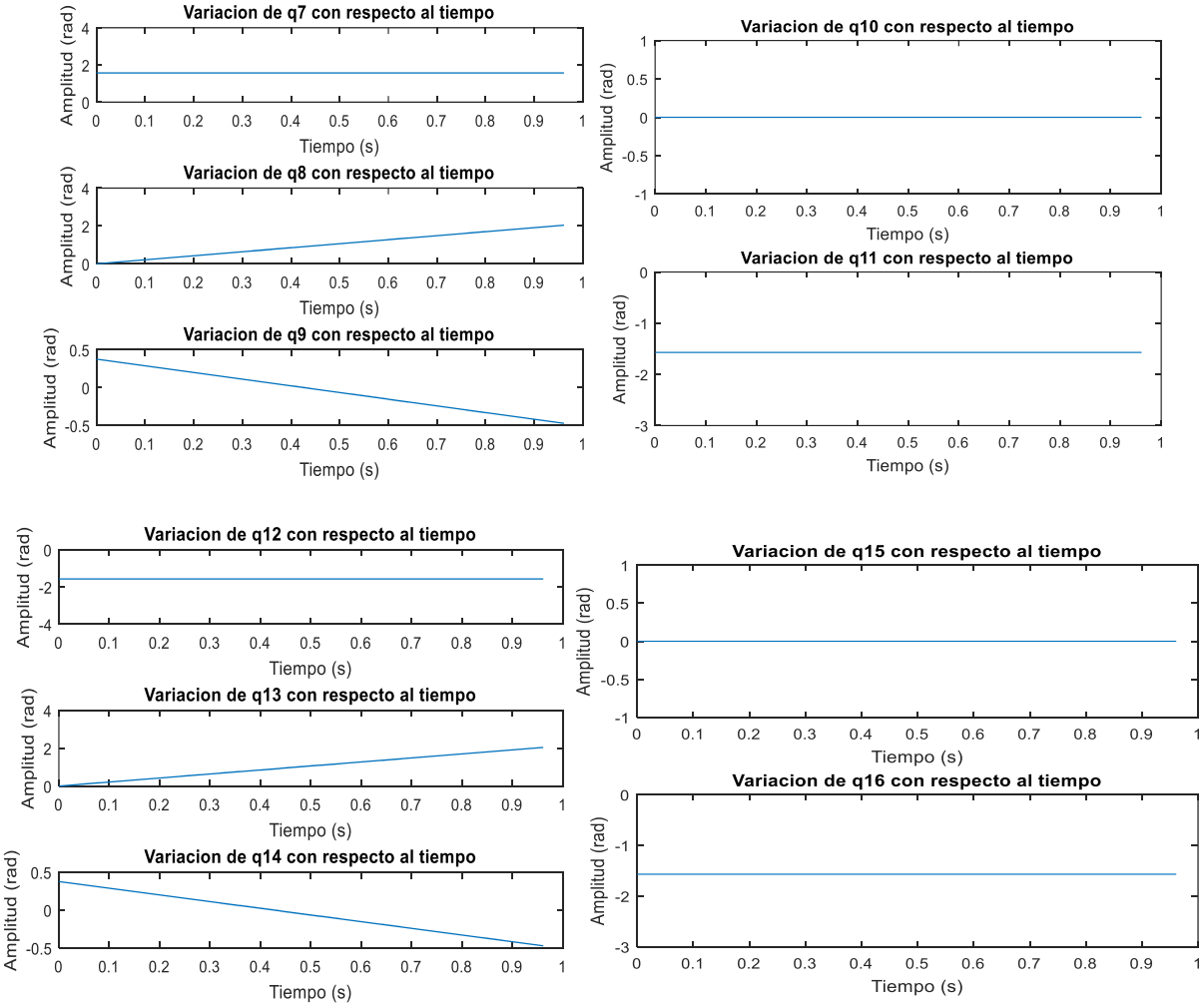


Figura 5.1.3.4. Variación de las articulaciones de las piernas

5.1.4. RANGO DE OPERACIÓN SIMULACIÓN.

El rango de operación es el intervalo que describe las capacidades de movimiento de cada articulación, este parámetro fue obtenido teniendo en cuenta las capacidades físicas del robot humanoide y las capacidades físicas del ser humano, respetando los límites que ambos tengan.

En las figuras 5.1.4.1., y 5.1.4.2., podemos observar el brazo izquierdo y brazo derecho de vista lateral, donde se encuentran la articulación q1 y q4 respectivamente, el rango de operación de cada articulación es de 0° a 180° en ambos casos.

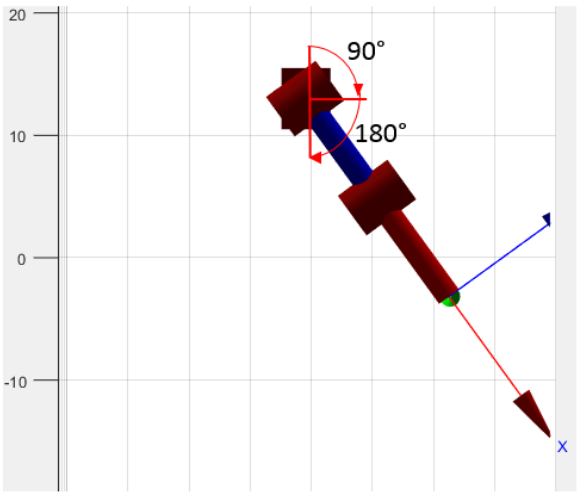


Figura 5.1.4.1. Rango de operación brazo izquierdo vista lateral.

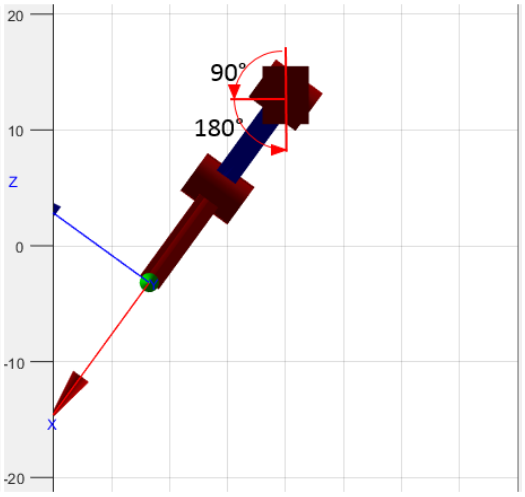


Figura 5.1.4.2. Rango de operación brazo derecho vista lateral.

En la figura 5.1.4.3., podemos observar el brazo izquierdo y brazo derecho de vista frontal, donde se encuentran las articulaciones q2, q3 y q5, q6 respectivamente, el rango de operación de las articulaciones q2 y q5, es de 0° a 180° y de las articulaciones q3 y q6, es de 0° a 90°.

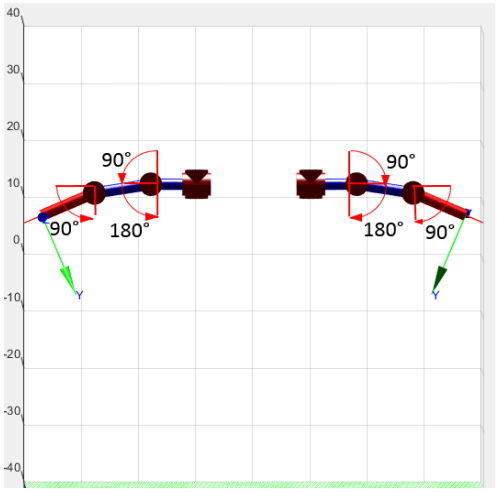


Figura 5.1.4.3 Rango de operación brazo izquierdo y derecho vista frontal.

En la figura 5.1.4.4., podemos observar la pierna izquierda de vista lateral y frontal, donde se encuentran las articulaciones q7, q8, q9, q10 y q11, el rango de operación de las articulaciones q7, q9 y q10 es de 0° a 180° y de las articulaciones q8 y q11, es de 0° a 90°.

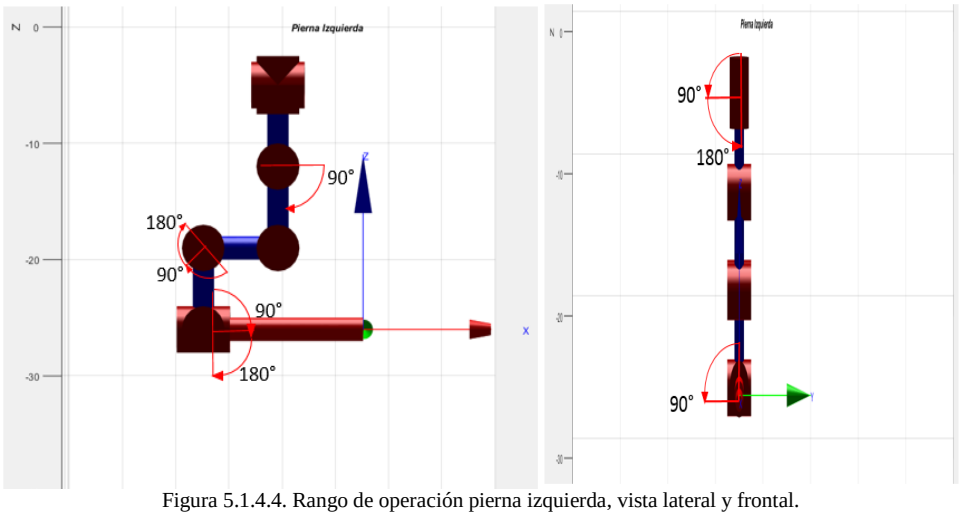


Figura 5.1.4.4. Rango de operación pierna izquierda, vista lateral y frontal.

En la figura 5.1.4.5., podemos observar la pierna izquierda de vista lateral y frontal, donde se encuentran las articulaciones q7, q8, q9, q10 y q11, el rango de operación de las articulaciones q12, q14 y q15 es de 0° a 180° y de las articulaciones q13 y q16, es de 0° a 90°.

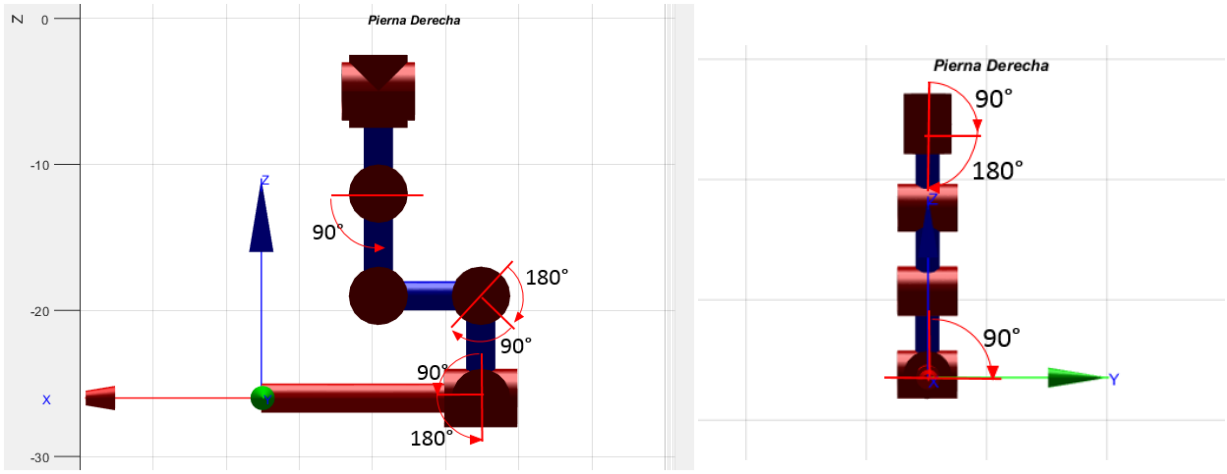


Figura 5.1.4.5. Rango de operación pierna derecha, vista lateral y frontal.

5.2. RESULTADOS EXPERIMENTALES.

Los parámetros más relevantes y de los cuales se podía llevar un análisis que demostrara el cumplimiento del objetivo de nuestro proyecto son: los errores cartesianos y articulares, rango de operación y el tiempo de respuesta.

El SDK que permite la manipulación e interacción del usuario con el humanoide no tiene compatibilidad con MATLAB, por esto no fue posible la intercomunicación del robot humanoide con MATLAB y debido a ello fue necesario utilizar un software compatible con el SDK, en nuestro caso utilizamos Visual Studio.

Para muchos de los resultados experimentales no se encontró una formulación matemática que evidenciara los datos que aquí se mostraran, en consecuencia se determinó que por análisis grafico se puede llegar a una conclusión respecto a los parámetros a analizar.

En los resultados experimentales fue posible realizar 4 de los 6 movimientos realizados en simulación, los movimientos que no se pudieron llevar a cabo con el robot humanoide fueron flexión de rodillas y

cuclillas, debido a, el SDK que suministra el fabricante posee funciones de seguridad y protección del robot humanoide, cuando la posición que sigue la trayectoria de la cadera del ser humano descienda más de 20 cm, deshabilita el humanoide de tal manera que finaliza de manera instantánea el seguimiento de trayectoria y reiniciando el humanoide para comenzar con una trayectoria nueva. Por consiguiente se intentó realizar el procedimiento de deshabilitar dichas funciones de seguridad que nos permitieran realizar todos los movimientos deseados.

5.2.1. ERROR CARTESIANO EN ESTADO ESTACIONARIO.

El error cartesiano en estado estacionario fue tomado mediante la comparación grafica de la posición final del actuador final del robot humanoide con la posición final de las extremidades del ser humano, debido a que la percepción del plano coordenado que se implantara en el humanoide cambia de una prueba a otra, en cambio al comparar las posiciones finales se puede tener una aproximación del error existente.

- Brazos cruzados al pecho.
  La vía por el cual fue analizado el error en estado estacionario fue gráficamente y comparando la posición de los eslabones que componen cada cadena cinemática. La plataforma del SDK del humanoide no permite extraer datos específicos del sensor visual RGB-D Kinect, a diferencia de MATLAB, por ende el método utilizado para encontrar el error fue comparar gráficamente la posición del usuario y la posición del robot humanoide, de esta manera se encuentra un error aproximado de la posición del humanoide.

  En la figura 5.2.1.1., podemos observar que el brazo izquierdo, tiene una posición con un error aproximado del 1% respecto a la imagen del usuario, por otra parte el brazo derecho tiene un error aproximado del 25% respecto a la imagen del usuario.

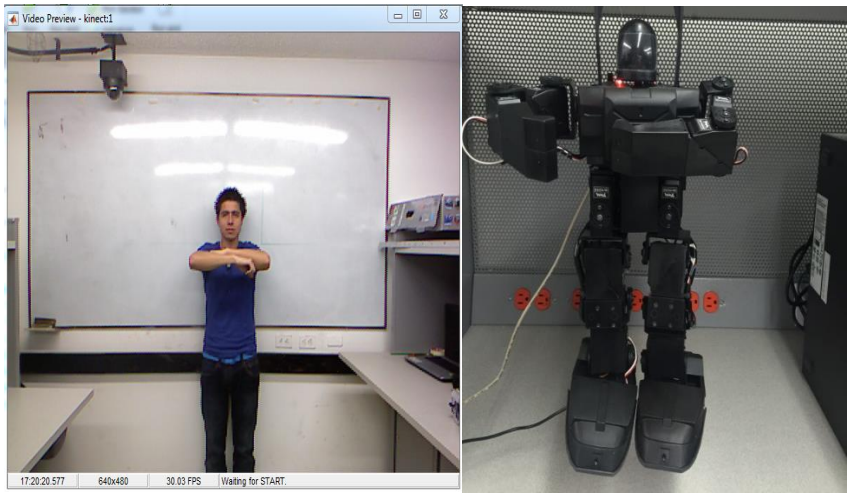


Figura 5.2.1.1. Brazos cruzados al pecho del usuario y el humanoide.

- Brazos completamente estirados hacia arriba.
  La vía por el cual fue analizado el error en estado estacionario fue gráficamente y comparando la posición de los eslabones que componen cada cadena cinemática. La plataforma del SDK del humanoide no permite extraer datos específicos del sensor visual RGB-D Kinect, a diferencia de MATLAB, por ende el método utilizado para encontrar el error fue comparar gráficamente la

posición del usuario y la posición del robot humanoide, de esta manera se encuentra un error aproximado de la posición del humanoide.

En la figura 5.2.1.2., podemos observar que el brazo izquierdo, tiene una posición con un error aproximado del 5% respecto a la imagen del usuario, por otra parte el brazo derecho tiene un error aproximado del 25% respecto a la imagen del usuario.

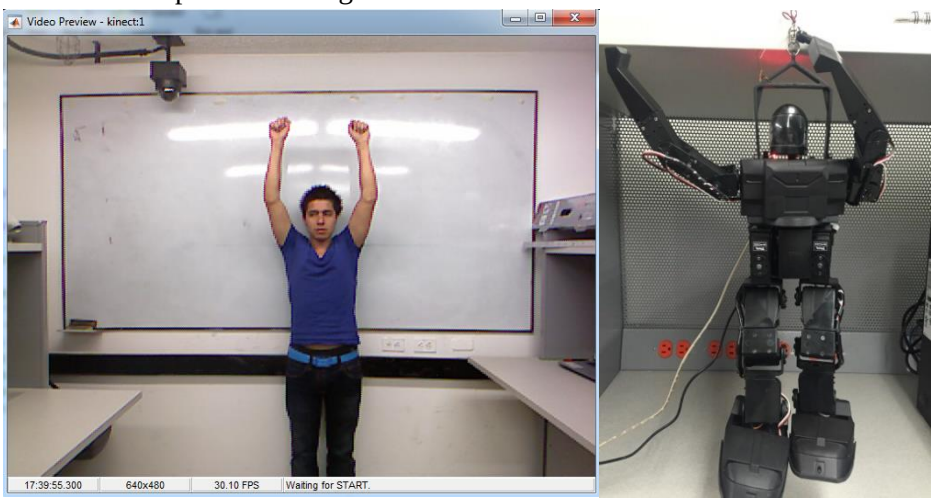


Figura 5.2.1.2. Brazos totalmente estirados hacia arriba del usuario y el humanoide.

- Brazos completamente estirados hacia los lados.
  La vía por el cual fue analizado el error en estado estacionario fue gráficamente y comparando la posición de los eslabones que componen cada cadena cinemática. La plataforma del SDK del humanoide no permite extraer datos específicos del sensor visual RGB-D Kinect, a diferencia de MATLAB, por ende el método utilizado para encontrar el error fue comparar gráficamente la posición del usuario y la posición del robot humanoide, de esta manera se encuentra un error aproximado de la posición del humanoide.

  En la figura 5.2.1.3., podemos observar que el brazo izquierdo, tiene una posición con un error aproximado del 5% respecto a la imagen del usuario, por otra parte el brazo derecho tiene un error aproximado del 5%, respecto a la imagen del usuario.

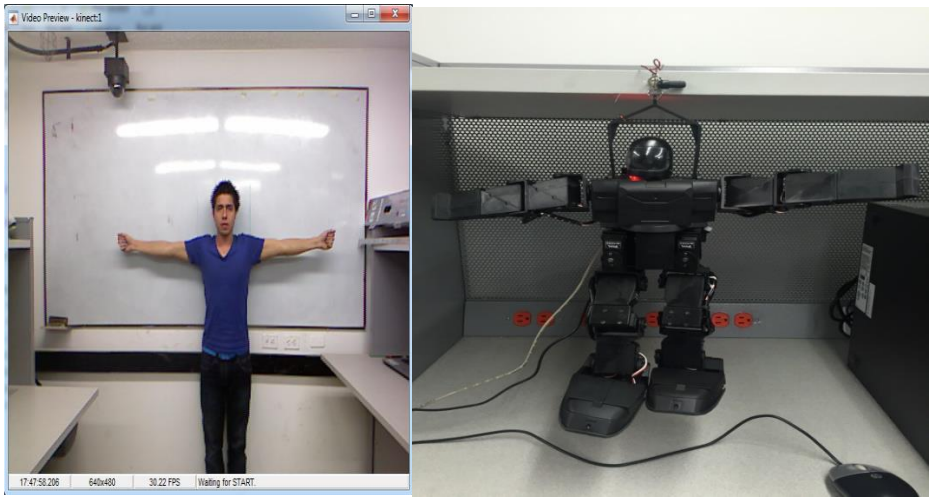


Figura 5.2.1.3. Brazos estirados hacia los lados del usuario y el humanoide.

5.2.2. ERROR ARTICULAR EN ESTADO ESTACIONARIO.

El error articular en estado estacionario fue tomado mediante la comparación de los valores de las articulaciones del robot humanoide con los valores de las articulaciones del cuerpo humano, tomando como referencia los valores de 0°- 90°-180° respectivamente, dado que son ángulos críticos donde se abarca la mayoría de los movimientos realizados, además de ser percibidos con mayor exactitud por el usuario. El robot humanoide cuenta con unas referencias donde se demarcan con exactitud los valores de 0°, 90° y 180°.

En el análisis de los errores experimentales se tuvo en cuenta el error absoluto y el error relativo, obtenidos de la siguiente manera:

$$e_{abs} = Valor\ medido - Valor\ de\ referencia$$

$$e_r = \frac{\sum_1^3 (Valor\ medido - Valor\ de\ referencia)}{\sum_1^3 Valor\ de\ referencia} * 100\%$$

De tal manera que nos permitiera calcular el error individual de las articulaciones y un error para la cadena cinemática total en cada movimiento, los resultados aquí mostrados dependen de la percepción del analista, por ende se pueden tomar como datos aproximados.

- Brazos cruzados al pecho. Articulación
  Se realizó la medición de los ángulos en estado estacionario del humanoide, obteniendo los siguientes resultados:

Datos / Articulación	Referencia	Medidas	Error Absoluto	Error Relativo
q1	90°	90°	0°	0%
q2	0°	30°	30°	16.66%
q3	90°	90°	0°	0%
q4	90°	90°	0°	0%
q5	0°	0°	0°	0%
q6	90°	90°	0°	0%

Tabla 5.2.2.1. Medición error absoluto y relativo brazos cruzados al pecho.

El error total del brazo derecho es de: 16.66%.
El error total del brazo izquierdo es de: 0% aprox.

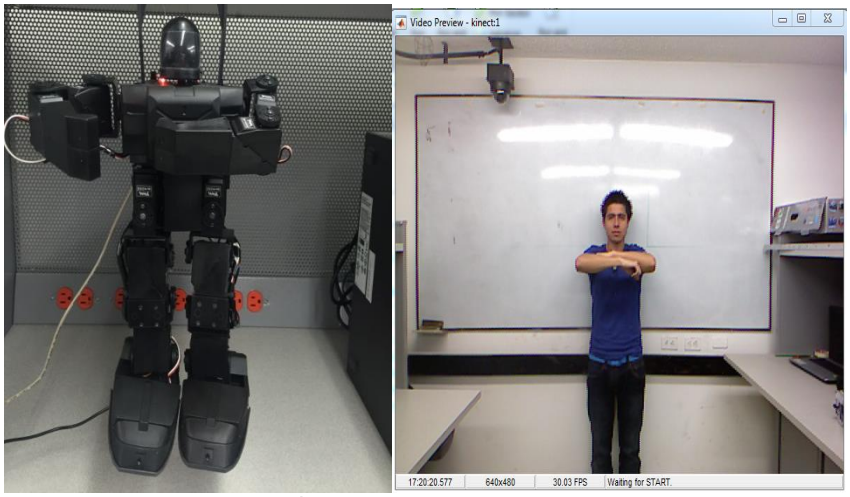


Figura 5.2.2.1. Medición error absoluto y relativo brazos cruzados al pecho.

- Brazos completamente estirados hacia arriba.
  Se realizó la medición de los ángulos en estado estacionario del humanoide, obteniendo los siguientes resultados:

Datos / Articulación	Referencia	Medidas	Error Absoluto	Error Relativo
q1	180°	180°	0°	0%
q2	0°	30°	30°	16.21%
q3	5°	5°	0°	0%
q4	180°	155°	-25°	-13.51%
q5	0°	0°	0°	0%
q6	10°	15°	-5°	-8.10%

Tabla 5.2.2.2. Medición error absoluto y relativo brazos completamente estirados hacia arriba.

El error total del brazo derecho es de: 16.21%.
El error total del brazo izquierdo es de: -21.61%.

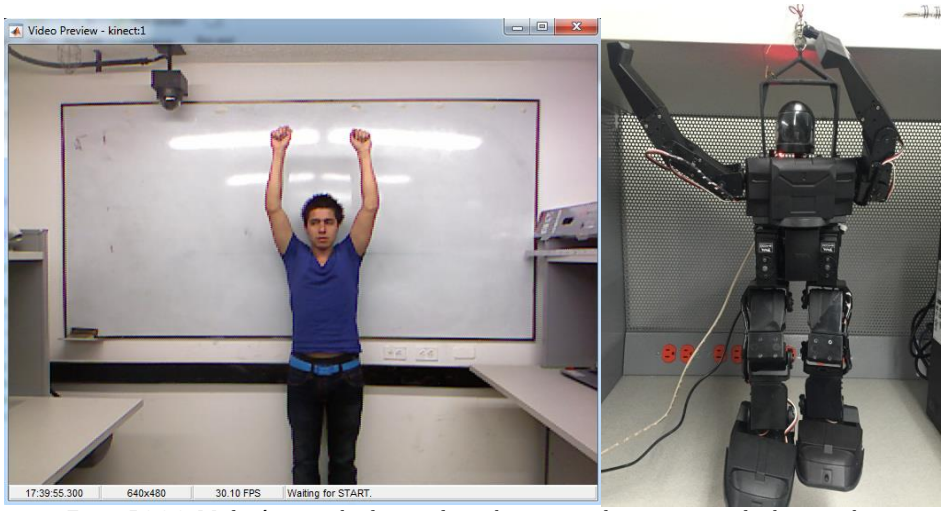


Figura 5.2.2.2. Medición error absoluto y relativo brazos completamente estirados hacia arriba.

- Brazos completamente estirados hacia los lados.
  Se realizó la medición de los ángulos en estado estacionario del humanoide, obteniendo los siguientes resultados:

Datos / Articulación	Referencia	Medidas	Error Absoluto	Error Relativo
q1	90°	90°	0°	0%
q2	90°	120°	30°	16.66%
q3	0°	0°	0°	0%
q4	90°	100°	10°	5.55%
q5	90°	45°	-45°	-25%
q6	0°	0°	0°	0%

Tabla 5.2.2.3. Medición error absoluto y relativo brazos completamente estirados hacia arriba.

El error total del brazo derecho es de: 16.66%.
El error total del brazo izquierdo es de: 30.55%.

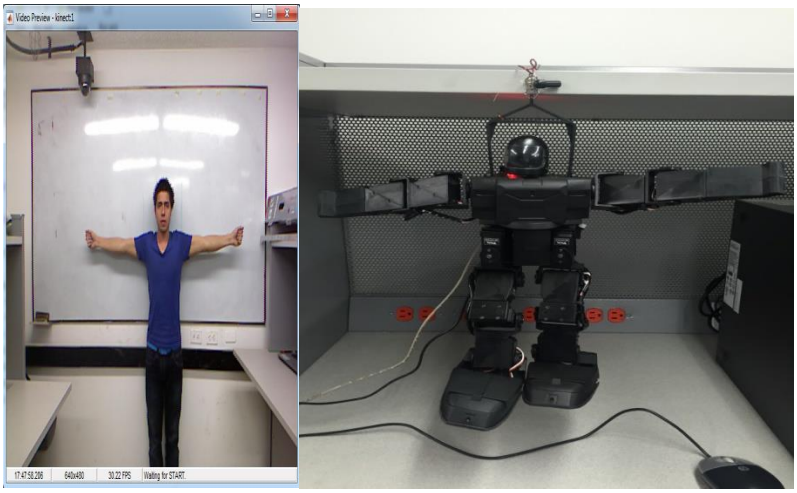


Figura 5.2.2.2. Medición error absoluto y relativo brazos completamente estirados hacia los lados.

5.2.3. RANGO DE OPERACIÓN ROBOT HUMANOIDE.

El rango de operación es el intervalo que describe las capacidades de movimiento de cada articulación, este parámetro fue obtenido teniendo en cuenta las capacidades físicas del robot humanoide y las capacidades físicas del ser humano, respetando los límites que ambos tengan.

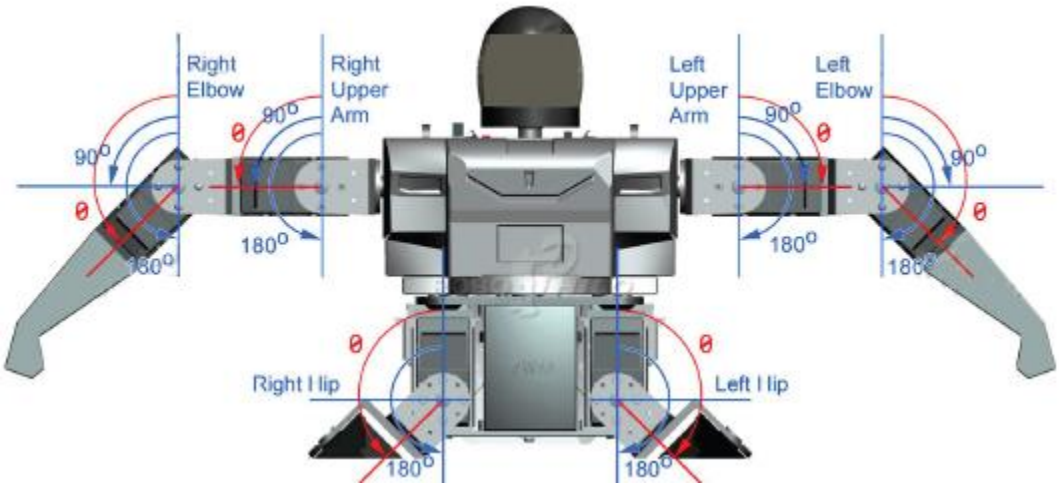


Figura 5.2.3.1. Rango de operación brazos y cadera del humanoide, vista frontal. [12]

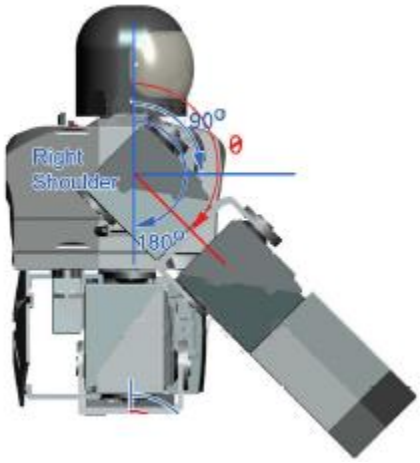


Figura 5.2.3.2.Rango de operación brazo derecho del humanoide, vista lateral. [12]

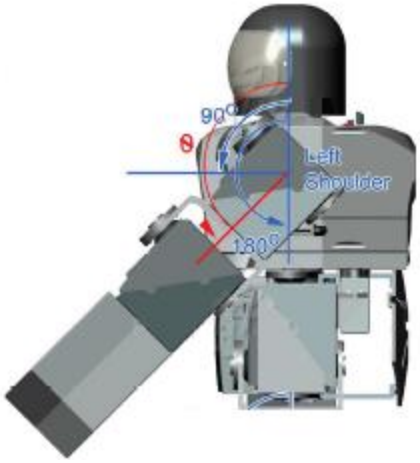


Figura 5.2.3.3. Rango de operación brazo izquierdo del humanoide, vista lateral. [12]

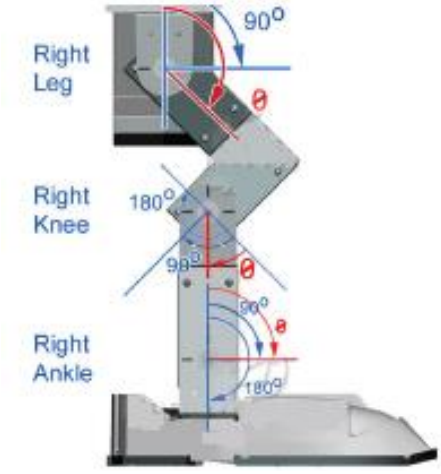


Figura 5.2.3.4. Rango de operación pierna derecha del humanoide, vista lateral. [12]

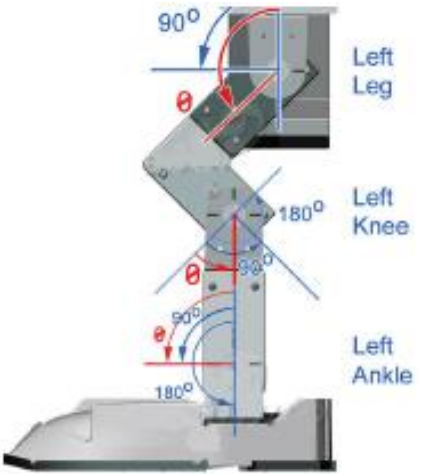


Figura 5.2.3.5. Rango de operación pierna izquierda del humanoide, vista lateral. [12]

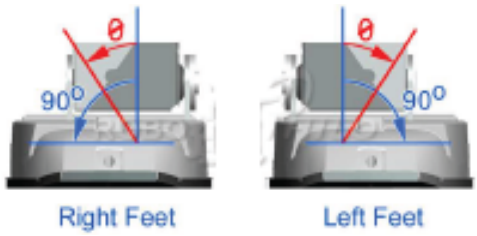


Figura 5.2.3.6. Rango de operación de los pies del humanoide, vista frontal. [12]

5.2.4. TIEMPO DE RESPUESTA.

El tiempo de respuesta nos permite ver la eficacia del control sobre el robot humanoide a determinados estímulos, para poder tener valor aproximado se realizó la tomada de datos de dos maneras: primero, se estableció que el usuario se movería lentamente, de esta manera se obtendrían una lista de datos, segundo, se estableció que el usuario se movería rápidamente, para obtener unos valores diferentes de respuesta.

Datos obtenidos a partir de un movimiento lento por parte del usuario:

Intento / Movimiento	Tiempo 1 (ms)	Tiempo 2 (ms)	Tiempo 3 (ms)	Tiempo 4 (ms)	Tiempo promedio (ms)
Brazos cruzados al pecho	1120	920	1100	940	1020
Brazos totalmente estirados hacia arriba	850	740	920	820	832.5
Brazos totalmente estirados hacia los lados	960	900	960	940	940

Tabla 5.2.4.1. Tiempo de respuesta para cada movimiento experimental.

6. CONCLUSIONES Y RECOMENDACIONES

El modelo teórico del robot humanoide que permitió el desarrollo de la simulación del robot humanoide puede llegar a, no tener solución, tener única solución o en definitiva múltiples soluciones, debido a ello es necesario el uso de funciones que cumplan con los requerimientos estipulados, que en este proyecto fue obtener una única solución y de esta manera no se obstruyera el procesamiento de los datos.

Es necesario un óptimo acople al cambio de plataforma para poder interactuar con el robot humanoide e implementar las funciones ya planteadas en simulación, de lo contrario el software interno del robot puede generar errores y no permitir un correcto funcionamiento del control, una evidencia de esto fue, no haber logrado el control de las extremidades inferiores con el robot humanoide, por otra parte evidenciando el correcto funcionamiento de las extremidades superiores del robot humanoide, nos da una guía del funcionamiento que podrían llegar a tener las extremidades inferiores cuando se pueda aplicar el control desarrollado.

Un criterio importante a analizar es el retardo que tiene el sistema para poder llevar acabo determinados movimientos, dicho retardo tiene varios factores por mejorar, como lo son, la velocidad de comunicación entre el robot y el computador con un valor de 9600 baudios por segundo y el microprocesador del robot humanoide permite comunicaciones más elevadas, esto ayudaría a mejorar el retardo; otro parámetro relevante es la complejidad del código, se puede llegar a una funcionalidad recursiva del mismo, permitiendo mejorar el tiempo de lectura y envió de resultados al robot, además el procesamiento de imágenes, toma un tiempo considerable y puede verse reflejado en la respuesta del sistema.

En los resultados mostrados en este proyecto se puede evidenciar, que la unificación de todos los componentes del desarrollados fue exitosa a pesar de ser desarrollados en distintas plataformas, se recomienda en trabajos futuros mejorar, la velocidad de comunicación, como se explicó anteriormente es un factor importante para la respuesta del sistema; el error en estado estacionario, establecer límites para no deteriorar el hardware del robot, además de cerciorarse del buen funcionamiento de cada una de sus partes.

Para mejorar la implementación de este proyecto sería necesario ir a fondo en la programación del mismo, logrando modificar funciones vitales de funcionamiento, para ello es necesario tener un conocimiento considerable acerca de programación; se recomienda aplicar este proyecto en otras plataformas robóticas con un nivel más avanzado en hardware y así evidenciar que tan efectivo es el control a nivel general y que relevancia tienen los cambios que se realicen.

7. BIBLIOGRAFÍA

[1]. LEE, JAEMIN; MANSARD, NICOLAS; PARK, JAEHEUNG. INTERMEDIATE DESIRED VALUE APPROACH FOR TASK TRANSITION OF ROBOTS IN KINEMATIC CONTROL. *ROBOTICS, IEEE TRANSACTIONS ON*, 2012, VOL. 28, NO 6, P. 1260-1277.

[2]. KOFMAN, JONATHAN, ET AL. TELEOPERATION OF A ROBOT MANIPULATOR USING A VISION-BASED HUMAN-ROBOT INTERFACE. *INDUSTRIAL ELECTRONICS, IEEE TRANSACTIONS ON*, 2005, VOL. 52, NO 5, P. 1206-1219.

[3]. KELLY, RAFAEL; SALGADO, RICARDO. PD CONTROL WITH COMPUTED FEEDFORWARD OF ROBOT MANIPULATORS: A DESIGN PROCEDURE. *ROBOTICS AND AUTOMATION, IEEE TRANSACTIONS ON*, 1994, VOL. 10, NO 4, P. 566-571.

[4]. FLOREZ, J; CALDERÓN, F. APLICACIONES DE LA CÁMARA RGB-D EN VISIÓN POR COMPUTADOR, XVI SIMPOSIO DE TRATAMIENTO DE SEÑALES, IMÁGENES Y VISIÓN ARTIFICIAL STSIVA 2011.

[5]. PRIMESENSE, "DISTANCE-VARYING ILLUMINATION AND IMAGING TECHNIQUES FOR DEPTH MAPPING," US, PATENT (US2010/0290698).

[6]. ANTONIO, B., FELIPE, P. L., CARLOS, B., & RAFAEL, A. (1997). FUNDAMENTOS DE ROBÓTICA. *ED. MC GRAW HILL*.

[7]. ROBOPHILO [EN LINEA]-URL: HTTP://WWW.ROBOPHILO.COM/LIVE/EN/#

[8]. CORKE, P. (2011). *ROBOTICS, VISION AND CONTROL: FUNDAMENTAL ALGORITHMS IN MATLAB* (VOL. 73). SPRINGER SCIENCE & BUSINESS MEDIA.

[9]. GUIA DEL USUARIO ROBOPHILO.

[10]. EJEMPLOS PARA GRAFICAR EL ESQUELETO EN MATLAB [EN LINEA]-URL: HTTP://WWW.MATHWORKS.COM/HELP/IMAQ/EXAMPLES/USING-THE-KINECT-R-FOR-WINDOWS-R-FROM-IMAGE-ACQUISITION-TOOLBOX-TM.HTML?REFRESH=TRUE

8. ANEXOS

Los anexos los encontrara en el siguiente link:

https://www.dropbox.com/home/Entrega%20Final%20Trabajo%20de%20grado%201517