UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE QUITO

CARRERA:

INGENIERÍA ELECTRÓNICA

Trabajo de titulación previo a la obtención del título de:

INGENIEROS ELECTRÓNICOS

TEMA:

TELEOPERACIÓN DEL ROBOT NAO PARA LA EJECUCIÓN DE TAREAS EN ESPACIOS REDUCIDOS USANDO EL DISPOSITIVO KINECT V2.

AUTORES:

DAVID SEBASTIÁN GALARZA GUERRERO

CHRISTIAN MAURICIO LLUMIQUINGA PUMISACHO

TUTOR:

VÍCTOR VINICIO TAPIA CALVOPIÑA

Quito, marzo del 2018

CESIÓN DE DERECHOS DE AUTOR

Nosotros David Sebastián Galarza Guerrero con cédula de identidad No 1723132021 y Christian Mauricio Llumiquinga Pumisacho con cédula de identidad No 1713437976, manifestamos nuestra voluntad y cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de titulación intitulado: "TELEOPERACIÓN DEL ROBOT NAO PARA LA EJECUCIÓN DE TAREAS EN ESPACIOS REDUCIDOS USANDO EL DISPOSITIVO KINECT V2." Mismo que ha sido desarrollado para optar por el título de Ingenieros Electrónicos en la Universidad Politécnica Salesiana. Quedando la universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la ley de la Propiedad Intelectual, en nuestra condición de autores reservamos los derechos morales de la obra antes citada. En concordancia. Suscribimos este documento en el momento que hacemos entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.

David Sebastián Galarza Guerrero
C.I. 1723132021

Christian Mauricio Llumiquinga Pumisacho
C.I. 1713437976

Quito, marzo del 2018

DECLARATORIA DE COAUTORÍA DEL DOCENTE TUTOR

Yo declaro que bajo mi dirección y asesoría fue desarrollado el Proyecto Técnico. "TELEOPERACIÓN DEL ROBOT NAO PARA LA EJECUCIÓN DE TAREAS EN ESPACIOS REDUCIDOS USANDO EL DISPOSITIVO KINECT V2".
Realizado por David Sebastián Galarza Guerreo y Christian Mauricio Llumiquinga Pumisacho.

Obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerados como trabajo final de titulación.

Quito, marzo del 2018

Víctor Vinicio Tapia Calvopiña
C.I. 1708547219

DEDICATORIA

Quiero dedicar este proyecto a toda mi familia y amigos en especial a mis padres Nancy y Fabian que con sus consejos y apoyo incondicional me han motivado para culminar esta etapa importante en mi vida y en el transcurso de toda mi carrera universitaria, sus enseñanzas y experiencias me han convertido en la persona que soy ahora, con defectos, pero también con cualidades y principio que es lo realmente importante en la vida.

David Sebastián Galarza Guerrero.

Estas cortas palabras son dedicadas a cada una de las personas que hicieron posible cumplir con una de mis metas, primero quiero Agradecer a Dios por darme la vida y una familia con principios y valores que en el transcurso de tiempo fueron madurando y convirtiéndome en una persona de bien, a mi Padre Jorge que con su carácter equilibrado siempre me hizo discernir lo bueno de lo malo, a mi madre Manuela que con su cariño incondicional siempre estuvo a mi lado, a mis hermanos Jorge Luis y Alex. A mi nueva familia que dios puso en mi camino Wilfrido y Alicia un ejemplo de personas que uno toma como modelo para su vida, a mis cuñados Edy, Nely, Xavier que fueron un apoyo muy importante para que esta meta se cumpla, a todos mis sobrinos en especial Abimael, Samira y Benjamín que siempre me sacaron una sonrisa en momentos de tristeza, al amor de mi vida Lupita mi compañera eterna que siempre está a mi lado apoyándome y luchando para alcanzar nuevas metas para poder cumplir todos nuestros sueños y ahora a la razón de vivir a la persona que llego a nuestras vidas a robar nuestros corazones y llenar de alegría cada mañana a mi querida hija Amanda, querida familia Dios les pague.

Christian Mauricio Llumiquinga Pumisacho.

AGRADECIMIENTO

Agradecemos a Dios por permitirnos realizar nuestros estudios de pregrado en la Universidad Politécnica Salesiana ya que con sabiduría escoge a los maestros que dejan huellas en nuestras vidas universitaria por el bagaje de conocimientos impartidos conjugado con la disciplina y responsabilidad que los caracteriza en formar buenos profesionales y personas de bien.

Agradecemos al Ing. Santiago Andrade que con su conocimiento nos supo guiar para la elaboración del proyecto.

A nuestro tutor ing. Vinicio Tapia que en el transcurso de la realización del proyecto entregó su confianza y apoyo necesario para culminar con éxito este proyecto.

David Sebastián Galarza Guerrero

Christian Mauricio Llumiquinga Pumisacho

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente proyecto técnico tiene como finalidad la teleoperación del Robot Nao para la ejecución de tareas en espacios reducidos usando el dispositivo Kinect v2. Se diseñó e implementó algoritmos para teleoperar el Robot Nao por medio de los sensores del Kinect v2. El sensor Kinect v2 es un dispositivo controlador de videojuego muy utilizado y desarrollado por Microsoft para la video consola Xbox 360 y desde junio del 2011 para computadoras a través de Windows, el dispositivo Kinect v2 permite la comunicación entre el usuario y la consola sin la necesidad de contacto físico con los diferentes sensores del Kinect v2, el sensor tiene una interfaz natural que reconoce al esqueleto del cuerpo humano y replica los movimientos en la interfaz sin la necesidad de algún dispositivo conectado físicamente en el usuario, además de poseer un sensor de reconocimiento de comandos de voz. Por medio del reconocimiento de los movimientos del cuerpo humano en el sensor Kinect v2 se realizó la comunicación entre el sensor Kinect v2 y el Robot Nao para que replique los movimientos del usuario y en algún momento el Robot Nao pueda realizar tareas que son muy peligrosas, de ese modo se puede precautelar la vida de la persona cuando se expone a tareas en espacios reducidos o en situaciones con eminente peligro, teniendo en cuenta los alcances y limitaciones que puede tener el Robot Nao.

ABSTRACT

The purpose of this technical project is the teleoperation of the Nao Robot to perform tasks in small spaces by using the Kinect v2 device. Some algorithms were designed and implemented to teleoperate the Nao Robot through the Kinect v2 sensors. The Kinect v2 sensor is a video game controller device widely used and developed by Microsoft for the Xbox 360 video console, since June 2011 for PC through Windows. The Kinect v2 allows to establish communication, with the Kinect v2 sensors, between the user and the console without any physical contact. The sensor has a natural interface that recognizes the skeleton of the human body and replicates the movements in the interface, without the need of any device physically connected to the user, it also has a sensor which is able to recognize voice commands. Through the recognition of the movements of the human body in the Kinect v2 sensor, we have created the communication between the Kinect v2 sensor and the Nao Robot to replicate the user's movements. Sometime in the future, the Nao Robot may be able to perform dangerous tasks, so that people are not exposed to perform risky tasks in small spaces taking into account its capacity and limitations of the Nao Robot.

INTRODUCCIÓN

Con el desarrollo de la tecnología y el avance en el área de la robótica y las telecomunicaciones se han perfeccionado e integrado distintos sistemas y dispositivos que permiten la interacción humano-robot en la ejecución de actividades donde el ser humano por su condición anatómica no puede realizarlas.

En la actualidad se presentan actividades que el ser humano no puede realizarlas en zonas y lugares con espacios reducidos incluso donde el ambiente puede estar contaminado, ante este problema se propone la creación de un sistema que replique los movimientos del ser humano por medio del dispositivo Kinect v2 para que estas acciones las ejecute el Robot Nao en el sitio remoto y así evitar exponer la vida de la persona que realiza esa labor.

Para la obtención de réplicas de movimientos del Robot Nao se van a desarrollar algoritmos que identifiquen los movimientos del ser humano a través del dispositivo Kinect v2, también se van a realizar pruebas de campo y simular entornos donde el Robot sea capaz de ejecutar actividades de maniobra y transporte.

El capítulo uno detalla el tema del proyecto, planteamiento del problema, justificación, objetivo general, objetivos e-específicos y los beneficiarios del proyecto.

El capítulo dos menciona los conceptos generales que conforman el proyecto, características técnicas de los dispositivos a utilizar y componentes tanto de software como hardware que mejor se adaptan a las necesidades del proyecto.

El capítulo tres muestra el desarrollo de los algoritmos que permiten la comunicación del dispositivo Kinect v2 con el Robot Nao y demás sensores, así como simulaciones que comprueben el correcto funcionamiento.

En el capítulo cuatro se describen las pruebas de campo para comprobar la réplica de los movimientos del ser humano y análisis de los resultados.

CAPÍTULO 1

ANTECEDENTES

1.1 Planteamiento del problema

En el año 2015 en la ciudad de Quito se suscitó un caso de dos obreros que fallecieron mientras realizaban tareas de mantenimiento en una alcantarilla, esto se produjo por la inhalación de gases tóxicos que los llevo a la pérdida de conocimiento y posteriormente a la muerte.

El problema por solucionar es la creación de un sistema para el control del Robot Nao mediante la réplica de movimientos desde un sitio remoto utilizando un computador y el sensor Kinect v2. El proyecto beneficiará al ser humano en labores de espacios reducidos y contaminados donde el entorno es inaccesible, evitando consecuencias que pueda afectar la integridad de las personas.

1.2 Justificación

Actualmente con el desarrollo de la tecnología y los avances en el área de la robótica y las telecomunicaciones se han perfeccionado e integrado distintas aplicaciones en sistemas y dispositivos que permiten la interacción humano-robot en la realización de actividades que comúnmente las realiza el ser humano y que pone en riesgo su integridad.

Se propone el desarrollo de un sistema que permita al Robot Nao ser controlado remotamente por el ser humano por réplicas de movimiento a través del dispositivo Kinect v2, realizar actividades en zonas de difícil acceso que presenten problemas ambientales como contaminación, gases nocivos, altas y bajas temperaturas y espacios reducidos para evitar así exponer la vida y problemas en la salud de la persona que realiza esta labor.

1.3 Objetivos

1.3.1 Objetivo general

Teleoperar el Robot Nao para la ejecución de tareas en espacios reducidos usando el dispositivo Kinect v2 como sensor.

1.3.2 Objetivos específicos

- Realizar la comunicación entre los sensores del Kinect v2 con el software de desarrollo para obtener los movimientos similares al del ser humano.
- Desarrollar un algoritmo que identifique los movimientos del ser humano utilizando el dispositivo Kinect v2, para que el Robot Nao replique los movimientos.
- Realizar pruebas de campo para comprobar los movimientos en espacios reducidos.
- Simular un entorno para que el Robot Nao realice actividades de maniobra y transporte.

1.4 Tema

Teleoperación del Robot Nao para la ejecución de tareas en espacios reducidos usando el dispositivo Kinect v2.

1.5 Beneficiarios de la propuesta

El presente proyecto técnico beneficiará a personas que realicen actividades de maniobra y transporte en sitios con espacios reducidos, permitirá controlar el Robot Nao remotamente por medio del dispositivo Kinect v2 basándonos en réplicas de movimiento.

También servirá como punto de partida para futuros desarrollos en el área de la robótica, con el rápido avance tecnológico se puedan crear sistemas más complejos que repliquen cada vez mejor los movimientos del ser humano en tareas peligrosas, lugares difíciles de acceder o que impliquen algún tipo de riesgo, beneficiando a toda la sociedad en general.

CAPÍTULO 2
FUNDAMENTACIÓN TEÓRICA

Este capítulo hace una descripción tanto de software como hardware de los dispositivos y equipos que van a ser utilizados en el desarrollo del proyecto.

2.1 Introducción a la Robótica

El tema de la robótica es relevante en el plan de estudio de ingeniería, debido a la capacidad de los robots para realizar trabajos repetitivos y peligrosos. Su intención es la de relevar a un trabajador humano de una labor aburrida, desagradable o demasiado precisa. Al contrario de lo que se piensa no es más rápido que los humanos en la mayoría de aplicaciones, pero es capaz de mantener su velocidad durante un largo periodo. (Kumar, 2010)

El robot se define de manera formal en la Organización Internacional para la Estandarización (ISO), como un manipulador multifuncional reprogramable, capaz de mover materiales, piezas, herramientas o dispositivos especiales, a través de movimientos variables programados, para el desempeño de tareas diversas. (Kumar, 2010)

Figura 2.1. Robot de opciones binarias

Robot de opciones binarias, (Gold Binary Robot, 2016)

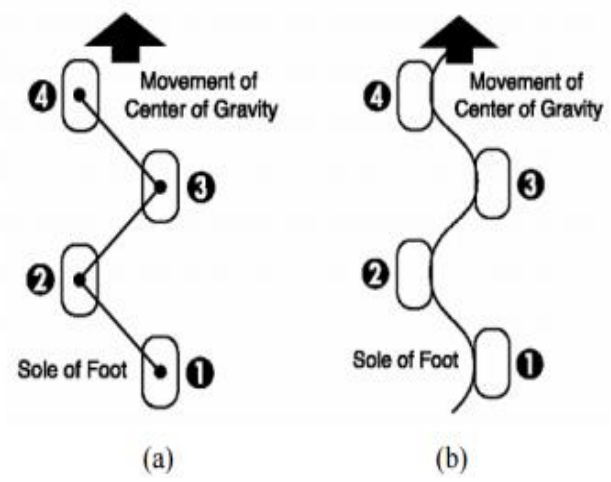
2.2 Robots Humanoides

Según las investigaciones en el desarrollo de la robótica se pretende construir cada vez más robots con características similares a los del ser humano. Los aspectos que se deben tomar en cuenta para lograr estos objetivos son la utilización de robots humanoides capaces de trabajar directamente en el mismo entorno que los humanos, en actividades de manipulación y movilización de objetos, utensilios etc. (Zoe Falormir, 2006)

Otro aspecto importante es que el medio de locomoción de los robots humanoides se pueda adaptar a su entorno, es decir, si el entorno en el que se quiere incorporar no contiene obstáculos y el suelo es liso, se puede utilizar un humanoide que se desplace en ruedas y si es un entorno real como (suelo desigual, escaleras, etc.) pueda utilizar un humanoide del tipo bípedo. (Zoe Falormir, 2006)

Se pueden distinguir dos tipos de locomoción bípeda: activa-estática que consiste en desplazamientos lentos donde el centro del movimiento permanece siempre en las plantas de los pies (se considera siempre estable), utiliza todos los grados de libertad del humanoide para realizar el movimiento, y pasiva-dinámica que consiste en movimientos rápidos al caminar, donde el centro de movimiento no se encuentra siempre centrado en las plantas de los pies (se considera inestable algunas veces), similar a la estrategia que utilizan los humanos para caminar. (Zoe Falormir, 2006)

Figura 2.2. Tipos de locomoción bípeda



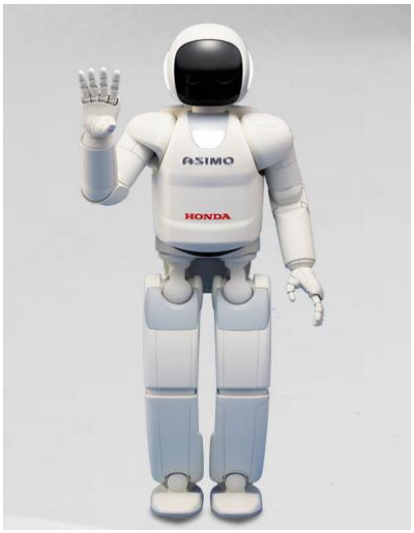
(a) Locomoción activa-estática y (b) Locomoción pasiva-dinámica. (Zoe Falormir, 2006)

2.2.1 Tipos de Robots Humanoides

Sin duda existe un sin número de robots que han sido diseñados para facilitar las tareas cotidianas del ser humano, a diferencia de otros con la capacidad de actuar y moverse exactamente igual que un ser humano para desenvolverse en el entorno y realizar distintas acciones, los denominados "robots humanoides'', punto en donde se centra el tema de estudio y donde se describe a continuación los más importantes hasta la actualidad.

a) **Asimo:** Desarrollado por la compañía japonesa Honda y considerado uno de los robots más complejos y avanzados del mundo por su capacidad de respuesta ante estímulos externos, alto nivel de equilibrio en sus extremidades y reconocimiento del entorno externo. (Honda, 2015)

Figura 2.3. Robot Asimo



Robot Asimo. (Honda, 2015)

b) **Atlas:** Este robot humanoide es considerado el mejor en actividades de rescate ya que puede manipular herramientas, evitar obstáculos y correr sin ningún problema. Fue creado en el año 2013 por la compañía Boston Dynamics con el financiamiento de Defense Advanced Research Projects Agency (DARPA) con el propósito de realizar acciones en caso de emergencia, situaciones de difícil acceso y condiciones extremas.

c) **Geminoid F:** Robot diseñado en Japón, está inspirado en una joven de 20 años y la que más se asemeja al aspecto físico de un ser humano, posee 12 grados de libertad y

es capaz de realizar 65 expresiones humanas, así como cantar y hablar, se espera que sea uno de los medios de comunicación más comunes para la vida cotidiana.

d) **Hubo:** Creado originalmente en el 2004 desde entonces con el avance de la tecnología ha sido mejorado para realizar actividades difíciles y supuestos escenarios catastróficos. (social, s.f.)

Figura 2.4. Robot Hubo

Robot Hubo. (social, s.f.)

e) **Nao:** Robot desarrollado por Aldeberan Robotics en el año 2005 como sustituto del perro robot de Sonic y con el fin de ser usado en labores de investigación y el área educativa en universidades y laboratorios. Fue diseñado para ser personalizado, cargar contenido y desarrollar nuevas habilidades según las necesidades del usuario. (Robotics A. , 2005)

Figura 2.5. Robot Nao

Robot Nao. (Robotics A. , 2005)

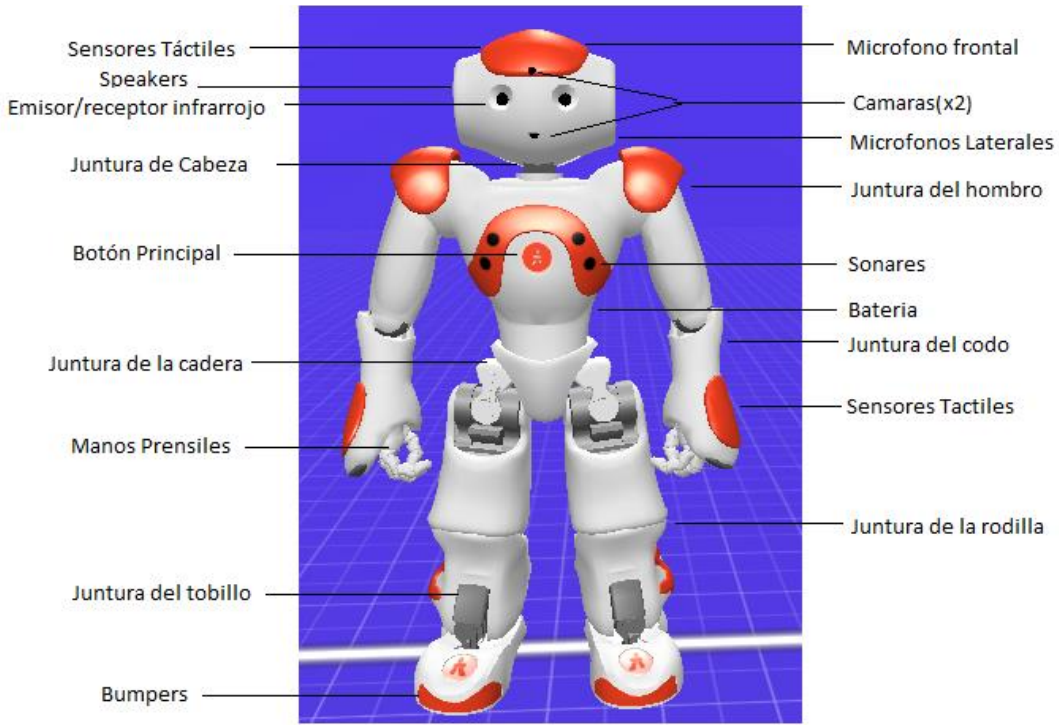
2.3 Robot Nao

Es un robot humanoide desarrollado por la empresa francesa Aldebaran Robotics, con el propósito de dar avances en el área de la robótica, asistencia a personas, y en la educación. Todo esto se logra a través de la plataforma de hardware y software que incorpora Nao.

2.3.1 Especificaciones Técnicas y Componentes

- Batería de Litio ion con una autonomía de aproximadamente 90 minutos.
- Mide 58 cm y pesa 4.8 kg.
- Memoria flash de 2GB.
- Procesador AMD GEODE x86 500MHz CPU con 256 MB de memoria SDRAM.
- Dos cámaras en la parte frontal para visión.
- Dos altavoces en las partes laterales de la cabeza (sintetizador de voz).
- Cuatro micrófonos en la cabeza, uno en la frontal, uno en la posterior, y uno a cada lado.
- Aproximadamente 26 Sensores y Actuadores (táctiles, ultrasonido, presión).
- 2 girómetros de un eje, un acelerómetro de 3 ejes y 2 sónares.

Figura 2.6. Componentes del Robot Nao



Componentes del Robot Nao.

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 2.6 muestra los componentes del robot Nao entre las cuales destacan los sensores, articulaciones, camaras y micrófonos para su interacción con el entorno.

Los grados de libertad del robot Nao para cada articulación hacen referencia a la libertad a través de la cual una medida o dato puede moverse y tomar diferentes valores a través de un espacio. La Tabla 2.1 muestra en detalle los valores para cada articulación.

Tabla 2.1. Grados de libertad de las articulaciones del Nao

Articulación	**Grados de Libertad**
Cabeza	2°
Brazos (c/u)	5°
Manos (c/u)	1°
Piernas (c/u)	5°
Pelvis	1°
Total	25°

Tabla con especificaciones de los grados de libertad del robot Nao

Elaborado por: David Galarza y Christian Llumiquinga

a) **Conectividad:** El robot Nao puede comunicarse a través de Wi-Fi protocolo 802.11g o Ethernet utilizando tecnología WPA (Wi-Fi Protected Access) y WEP (Wired Equivalent Privacy).

El protocolo 802.11g, así como todos sus antecesores, permite establecer conexiones inalámbricas a través de ondas electromagnéticas, en la capa física para la codificación de información y la capa de enlace de datos para el control de enlace lógico(LLC) y control de acceso al medio (MAC).

Brinda un ancho de banda de 54 Mbps a su capacidad máxima, llega hasta 30 Mbps en la práctica, el rango de frecuencia es de 2.4 GHz y al ser un protocolo mejorado de estándares anteriores (802.11a, 802.11b, 802.11c, 802.11d, 802.11e, 802.11f), es compatible con dispositivos que admitan dichos estándares.

b) Detección: Identificación del entorno y de personas a través de las cámaras.

c) Lenguaje: Reconocimiento de sonidos y hasta 19 idiomas distintos a través de cuatro micrófonos que incorpora.

d) Reconocimiento: Es capaz de detectar estados de ánimo y memorizar emociones tras considerar las expresiones de la persona.

e) Manos Prensiles: Es capaz de levantar hasta 600 gramos ya que cuenta con tres dedos flexibles.

f) Bumpers: Ubicados en los pies lo cual le permite detectar obstáculos a su paso. (Robotrónica, 2016)

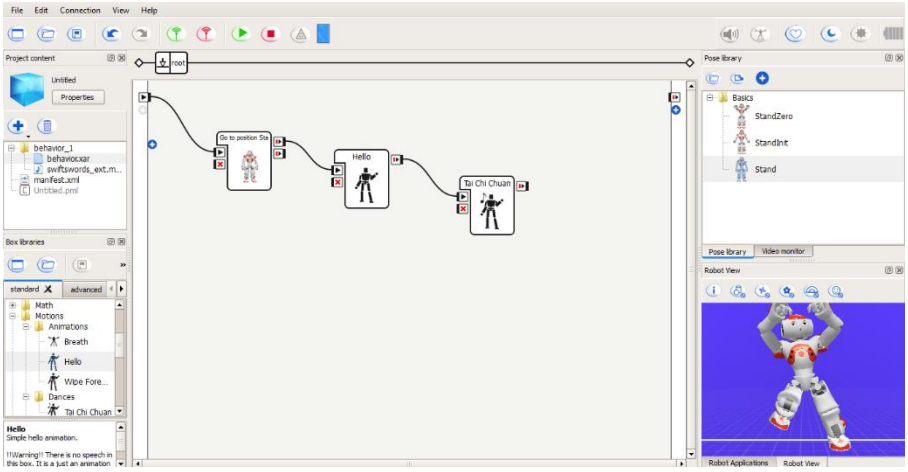
2.3.2 Software de programación del robot Nao

Con el propósito de facilitar la interacción con el robot Aldebaran Robotics ha diseñado su propio sistema operativo denominado "NAOqi OS", en conjunto con sistemas de programación que permitan un desarrollo a un nivel más avanzado. Se combinan distintas herramientas para acceder a los datos obtenidos de los sensores para crear funciones que permitan al robot ejecutar actividades de maniobra, transporte de objetos, hablar distintos idiomas, reconocimiento de voz, procesamiento de imágenes, reconocimiento de objetos entre otras actividades más sofisticadas. Choregraphe es software de programación desarrollado por Aldebaran Robotics con el fin de crear movimientos y comportamientos

para el robot Nao de manera práctica y sencilla ya que utiliza una interfaz gráfica a través de diagramas de flujo encadenando bloques o cajas uno tras otro los cuales representan bibliotecas con distintos comportamientos programados en lenguaje de programación Python, que pueden ser a su vez modificados y utilizados en secuencias lógicas.

Otros lenguajes como C++ también pueden ser utilizados a través de módulos desarrollados por separado.

Figura 2.7. Interfaz de Choregraphe



Secuencia de movimientos del Nao en Choregraphe.

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 2.7 muestra un diagrama de bloques realizado en Choregraphe para la realización de rutinas por parte del robot Nao.

a) Python

Es un lenguaje de programación con una estructura eficiente y de alto nivel orientado a objetos con una sintaxis simple y dinámica, usado en la mayoría de plataformas.

Conserva una extensa variedad de bibliotecas estándar que están a libre disposición en forma binaria y de código fuente en diversas plataformas en conjunto con el intérprete de Python en su sitio web oficial y distribuye su contenido libremente como programas, herramientas y documentación adicional. El intérprete de Python puede desarrollar nuevas funcionalidades implementados en C o C++ como un lenguaje de extensiones para aplicaciones personalizables. A continuación, se describen algunas ventajas de

Python sobre otros lenguajes de programación, no es necesario declarar variables ni argumentos, los tipos de datos permiten expresar operaciones complejas en una sola instrucción, las instrucciones se hacen por sangría en lugar de llaves. (Rossum, 2009).

b) C# Sharp

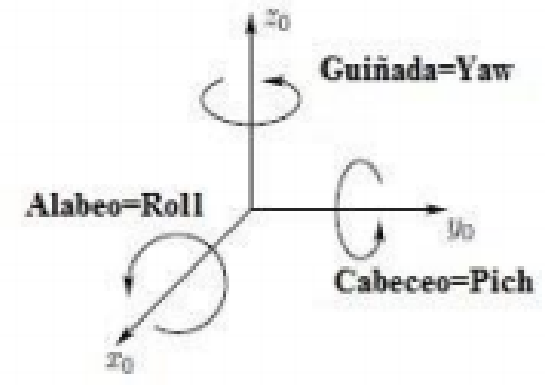
C Sharp es un tipo de lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma.NET, que después fue aprobado como un estándar por la European Computer Manufacturers Association (ECMA) y la Organización Internacional para la Estandarización (ISO).

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes.
El 19 de noviembre de 2007 salió la versión 3.0 de C# destacando entre las mejoras los tipos implícitos, tipos anónimos y LINQ (Language Integrated Query -consulta integrada en el lenguaje). (EcuRed, 2017)

2.3.3 Cinematica del robot Nao

Para describir los movimientos de un robot es necesario aplicar un modelo matemático empleando la matriz de rotación, la cual es el resultado del producto de rotaciones en los ejes (x, y, z) representados en el plano tridimensional como se muestra en la Figura 2.8.

Figura 2.8. Sistema de coordenadas tridimensionales.



Ángulos en plano tridimensional (Spong, 2006)

La matriz de transformación viene dada por las Ecuaciónes 2.1 y 2.2
Donde S=Sin y C=Cos. (Spong, 2006)

$$Rxyz = R_{z,\phi} R_{y,\theta} R_{x,\psi} \quad \text{Ec (2.1)}$$

$$Rxyz = \begin{bmatrix} C\phi & S\phi & 0 & 0 \\ S\phi & C\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C\theta & 0 & S\theta & 0 \\ 0 & 1 & 0 & 0 \\ -S\theta & 0 & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\psi & -S\psi & 0 \\ 0 & S\psi & C\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Ec (2.2)}$$

$$Rxyz = \begin{bmatrix} C\phi C\theta & -S\phi C\psi + C\phi S\theta S\psi & S\theta S\psi + C\phi S\theta S\psi & 0 \\ S\phi C\theta & C\phi C\psi + S\phi S\theta S\psi & -C\theta S\psi + S\phi S\theta C\psi & 0 \\ -S\theta & C\theta S\psi & C\theta C\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Ec (2.3)}$$

La Ecuación 2.3 expresa el resultado del producto de las tres matrices anteriores. (Spong, 2006)

La Ecuación 2.4 muestra la matriz de transformación para evaluar la cinemática específica para un modelo de robot humanoide. (Kofinas, 2012)

$$D = \begin{bmatrix} C\phi C\theta & -S\phi C\psi + C\phi S\theta S\psi & S\theta S\psi + C\phi S\theta S\psi & Px \\ S\phi C\theta & C\phi C\psi + S\phi S\theta S\psi & -C\theta S\psi + S\phi S\theta C\psi & Py \\ -S\theta & C\theta S\psi & C\theta C\psi & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Ec. (2.4)}$$

La cinemática del robot trata de movimientos dinámicos complejos que puede ejecutar el robot dependiendo del grado de liberad que le da cada articulación en un sistema de coordenadas fijas, el cual define los alcances máximos y mínimos que puede tomar las articulaciones al momento de desplazarse tomando como referencia los ángulos que forma por las distintas posiciones generadas por el robot. Para resolver este problema se analizará la cinemática directa y cinemática inversa.

a) Cinemática directa

Para resolver el problema de la cinemática directa se utiliza cálculos matemáticos como algebra vectorial y matricial esto ayudará a describir y representar cada uno de los

distintos componentes que conforman el robot en un espacio de tres coordenadas (x, y, z), cada componente del robot es un objeto firme unidos entre sí por articulaciones para formar el cuerpo del robot

b) Cinemática inversa

El método de la cinemática inversa nos ayuda a encontrar los movimientos que pueda realizar cada una de las partes del robot dependiendo del grado de liberad que tenga las articulaciones que conecta cada uno de los componentes.

Método Geométrico

Para la resolución del método de cinemática inversa se tomó como referencia el método geométrico ya que por tener 25 grados de liberad (DOF), el robot Nao se ajusta a los requerimientos, para la resolución por método geométrico es necesario conocer que el robot Nao está comprendido por cinco cadenas cinemáticas que son: la cabeza, dos brazos dos piernas con sus respectivas articulaciones que son las siguientes:

Tabla 2.2. Cadenas cinemáticas

CABEZA	Guiño de cabeza	Cabeceo				
BRAZO IZQUIERDO	Cabeceo hombro izquierdo	Alabeo hombro izquierdo	Guiñada codo izquierdo	Alabeo codo izquierdo	Guiñada muñeca izquierda	Mano izquierda
BRAZO DERECHO	Cabeceo hombro derecho	Alabeo hombro derecho	Guiñada codo derecho	Alabeo codo derecho	Guiñada muñeca derecha	Mano derecha
PIERNA IZQUIERDA	Cabeceo Guiñada de cadera	Alabeo cadera izquierda	Cabeceo cadera izquierda	Cabeceo rodilla izquierda	Cabeceo tobillo izquierdo	Alabeo tobillo izquierdo
PIERNA DERECHA		Alabeo cadera derecha	Cabeceo cadera derecha	Cabeceo rodilla derecha	Cabeceo tobillo derecho	Alabeo tobillo derecho

Nombres de articulaciones del robot Nao

Elaborado por: David Galarza y Christian Llumiquinga

La Tabla 2.2 muestra las cadenas cinematicas que posee el robot Nao para cada una de sus extremidades.

En general, se supone que el robot es completamente simétrico, pero curiosamente, según el fabricante, algunas articulaciones en el lado izquierdo tienen un rango diferente al de las articulaciones correspondientes en el lado derecho. Además, aunque algunas articulaciones parecen poder moverse dentro de un amplio rango, el controlador de hardware del robot prohíbe el acceso a los extremos de estos rangos, debido a posibles colisiones entre la carcasa que recubre al NAO. (Kofinas N. , 2012)

Tabla 2.3. Grados de libertad del Robot Nao

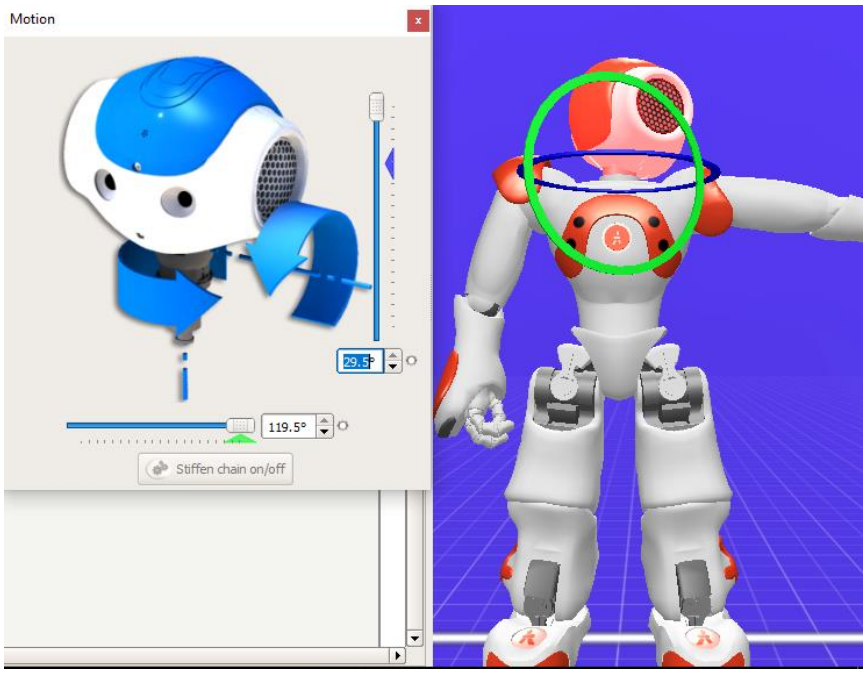
	ANGULOS EN GRADOS CENTIGRADOS	ANGULOS EN RADIANES
Cabeceo hombro izquierdo	-119.5° - 119.5°	-2.0857 - 2.0857
Alabeo hombro izquierdo	-18° - 76°	-0.3142 - 1.3265
Guiñada codo izquierdo	-119.5° - 119.5°	1.5446 - 0.0349
Alabeo codo izquierdo	-88.5° - -2°	-0.6720 - 0.5149
Guiñada muñeca izquierda	-104.5°- 104.5°	-1.8238 - 1.8238
Mano izquierda	0 – 1 (Lógicos)	No aplica
Cabeceo hombro derecho	-119.5° - 119.5°	-2.0857 - 2.0857
Alabeo hombro derecho	-38.5° - 29.5°	-1.3265 - 0.3142
Guiñada codo derecho	-119.5° - 119.5°	-2.0857 - 2.0857
Alabeo codo derecho	-38.5° - 29.5°	0.0349 - 1.5446
Guiñada muñeca derecha	-104.5°- 104.5°	-1.8238 - 1.8238
Mano derecha	0 – 1 (Lógicos)	No aplica

Análisis de valores máximos y mínimos que forman los ángulos de las extremidades

Elaborado por: David Galarza y Christian Llumiquinga

a) Cabeza: En la Figura 2.9 y Figura 2.10 se observa los ángulos máximos y mínimos que puede formar el Robot con respecto al sistema de referencia en tres dimensiones.

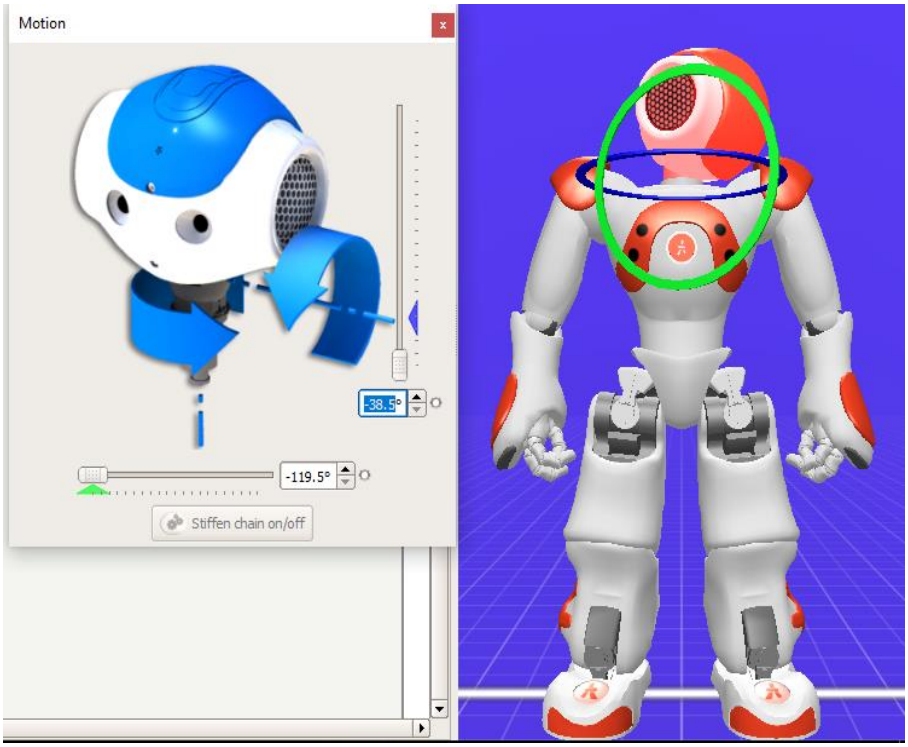
Figura 2.9. Grados de libertad de la cabeza



Valores máximos de los grados de libertad de articulación de la cabeza

Fuente: (Robotics C. , 2014)

Figura 2.10. Grado de libertad de la cabeza



Valores mínimos de los grados de libertad de articulación de la cabeza

Fuente: (Robotics C. , 2014)

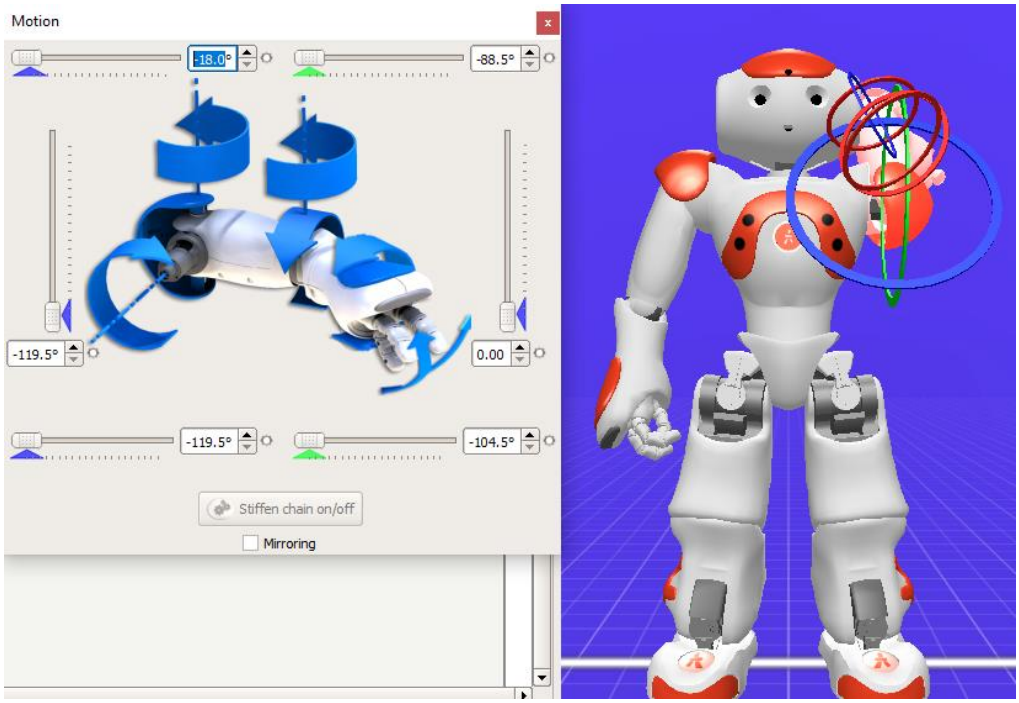
Para representar el movimiento de la cabeza como se puede observar en la Figura 2.8 y Figura 2.9 se tiene movimientos en el eje z y movimientos en el eje x, para ello se observa las siguientes ecuaciones:

$$\theta_1(eje\ z) = \left[\pm acos\left(\frac{\rho x}{L2COS\left(\theta_2-\frac{\pi}{2}\right)-L1SEN\left(\theta_2-\frac{\pi}{2}\right)}\right)\right] \quad \text{Ec. (2.5)}$$

$$\theta_2(eje\ x) = \left[\pi - asin\left(\frac{-\rho x+L3}{\sqrt{L1^2+L2^2}}\right) - atan\left(\frac{L1}{L2}\right)+\frac{\pi}{2}\right] \quad \text{Ec. (2.6)}$$

b) Brazo Izquierdo: En la Figura 2.11 y Figura 2.12 se observa los grados máximos y mínimos de los ángulos que pueden tomar las posiciones de cada articulación.

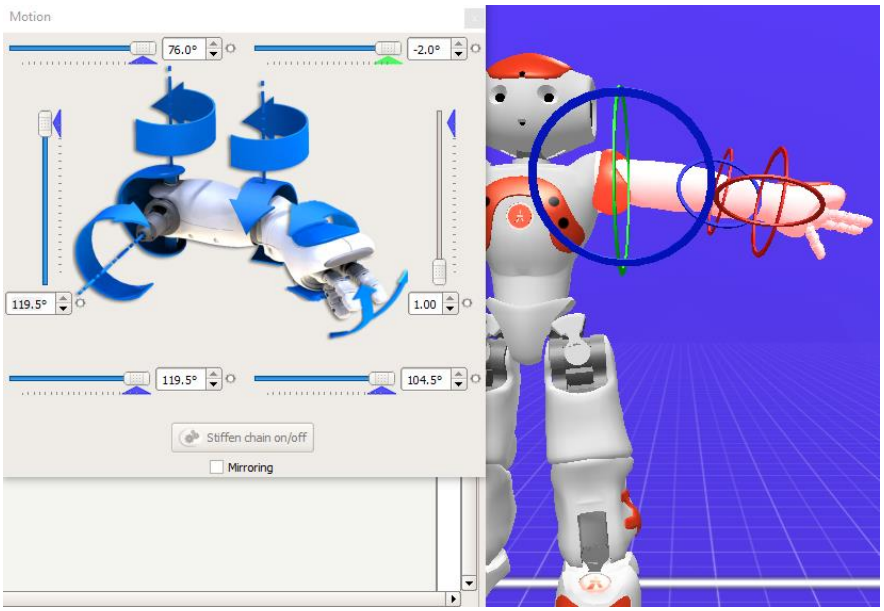
Figura 2.11. Grados de libertad del brazo izquierdo



Valores mínimos de grados de libertad de las articulaciones del brazo izquierdo

Fuente: (Robotics C. , 2014)

Figura 2.12. Grados de libertad del brazo izquierdo



Valores máximos de grados de libertad de las articulaciones del brazo izquierdo

Fuente: (Robotics C. , 2014)

Para el movimiento del hombro en la coordenada z se la representa por la ecuación 2.7.

$$D = A^0_{BASE} D^1_0 D^2_1 D^2_1 D^3_2 D^4_3 R_Z\left(\frac{\pi}{2}\right) A^{END}_4 \pi r^2 \qquad \text{EC. (2.7)}$$

Para el movimiento del hombro en la coordenada x se la representa por la ecuación 2.8.

$$D' = (A^0_{BASE})^{-1}\, D (A^{END}_4)^{-1} \left(R_Z\left(\frac{\pi}{2}\right)\right)^{-1} \qquad \text{EC. (2.8)}$$

Para el desplazamiento de la muñeca se asigna la siguiente Ecuación 2.9. (Kofinas, 2012)

$$D'' = (D')^{-1} \qquad \text{EC. (2.9)}$$

Los movimientos que puede realizar el codo son: LElbowYaw, LElbowRoll que se describe a continuación en las ecuaciones 2.10 y 2.11.

$$\theta_3 = \begin{Bmatrix} arcsin\left(\frac{D''_{3,4}}{L1}\right) \\ \pi - arcsin\left(\frac{D''_{3,4}}{L1}\right) \end{Bmatrix} \quad \text{EC. (2.10)}$$

$$\theta_4 = \pm acos\left(\frac{L2D''_{2,4} - L1D''_{1,4}cos\theta_3}{L2^2 + L1^2 cos^2\theta_3}\right) \quad \text{EC. (2.11)}$$

c) Brazo derecho: El brazo derecho e izquierdo son simétricos por lo tanto las ecuaciones son las mismas que las del brazo izquierdo.

2.4 Kinect

Kinect es un dispositivo creado en el año 2009 por la empresa Microsoft la cual anuncio su lanzamiento oficial en el año 2010, se origina con el propósito de controlar consolas de video sin necesidad de un mando tradicional sino a través de movimientos del cuerpo o mediante ordenes de voz que captan los distintos sensores de los que está compuesto el dispositivo. Esto se conoce como interfaz natural de usuario (Natural User Interface), con sus siglas NUI lo cual permite la interacción con sistemas o aplicaciones sin la necesidad de usar dispositivos cuyo funcionamiento sea aprendido.

Figura 2.13. Interfaz Natural de Usuario (NUI)

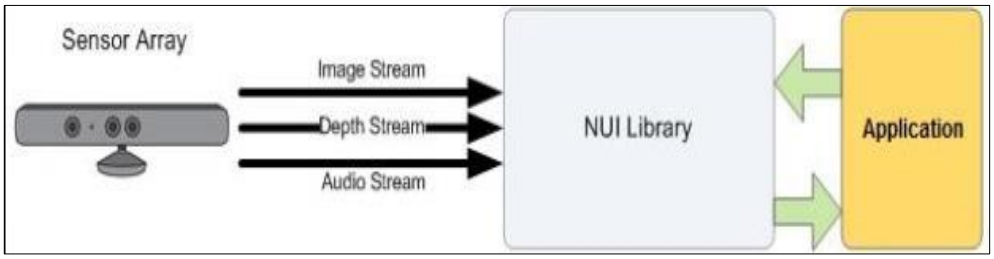


Diagrama de la interfaz natural de usuario de Kinect. (Kowalczyk, 2011)

La Figura 2.13 muestra el proceso de intercambio de información entre las aplicaciones y el sensor Kinect. Hasta el momento existen dos versiones de Kinect para Microsoft las cuales se detallarán a continuación.

2.5 Kinect V1

Primera versión desarrollada por Microsoft por Alex Kipman, se basa en Software creado por Rare, compañía perteneciente a Microsoft. Este dispositivo fue utilizado en principio para la consola de videojuegos Xbox 360, pero luego se hicieron desarrollos que permitieron crear controladores libres y de código abierto para usarlo como herramienta para aplicaciones en otras áreas de investigación. Esta tecnología cuenta con tres tipos de sensores, una cámara RGB, sensor de profundidad y micrófono multiarray que trabajan en conjunto para crear una interfaz de usuario natural, cada uno con características que fueron mejoradas con el desarrollo de la otra versión de Kinect. A continuación, se detalla las partes que conforman la primera versión del dispositivo.

Figura 2.14. Partes de Kinect V1

Partes de Kinect V1. (Sanford, 2012)

2.6 Kinect V2

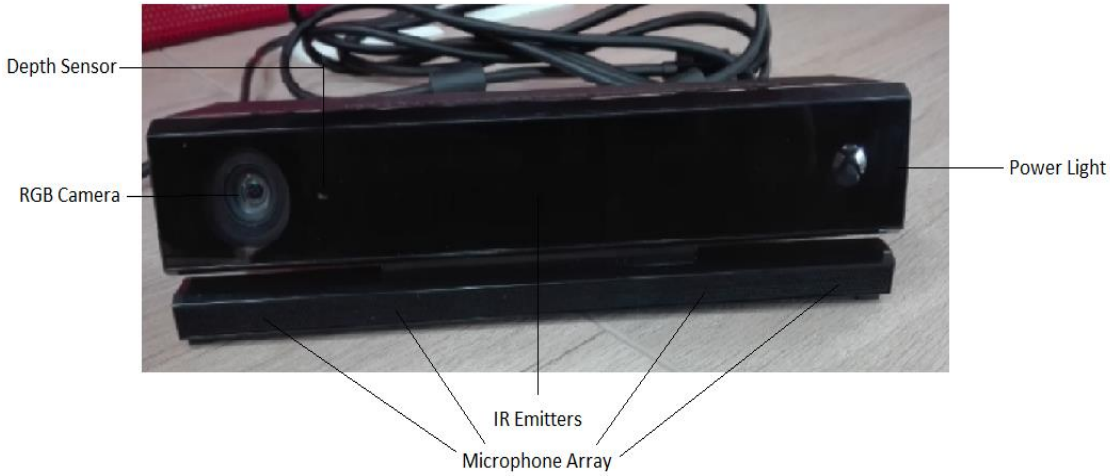
Nace como mejora de la primera versión de Kinect de Microsoft y en conjunto con la nueva consola de videojuegos de Xbox One en el año 2013, incorpora una mejora en los sensores que permiten la interacción con la persona, y al igual que con la versión anterior permite su desarrollo a través de código abierto disponible en varias plataformas y con distintas herramientas.

2.6.1 Características

Actualmente, Kinect V2 incorpora grandes avances en sus sensores que permiten una mejora en cuanto a calidad de imagen, detección de personas, captación del entorno y profundidad.

Destacan su mejora en la calidad y fidelidad de la cámara de profundidad, IR independiente que permite tener una nueva fuente para manipular, rastreo de la posición de la mano (abierta, cerrada y lazo), y crear varias aplicaciones a la vez. (Duque, 2015)

Figura 2.15. Componentes del dispositivo Kinect V2



Partes que conforman el dispositivo Kinect V2.
Elaborado por: David Galarza y Christian Llumiquinga

En la Tabla 2.4 se presentan las características técnicas entre el dispositivo Kinect V1 y Kinect V2.

Tabla 2.4. Características de los dispositivos Kinects

Capacidad	Kinect V1	Kinect V2
Profundidad	Resolución 320x240 0.8 a 4 metros de distancia	Resolución 512x424 0.5 a 4.5 metros de distancia
Rastreo de movimiento	Puede captar hasta 6 personas, pero solo rastrearlas	Puede identificar y detectar hasta 6 cuerpos completos
Video	640x480 30fps (colorstream)	1920x1080 30fps (colosource)
Puerto	USB 2.0	USB 3.0
Motor de inclinación	Si	No (Puede graduarse manualmente)

Tabla con detalles técnicos de las versiones de Kinect de Microsoft

Elaborado por: David Galarza y Christian Llumiquinga

2.6.2 Requerimientos Técnicos

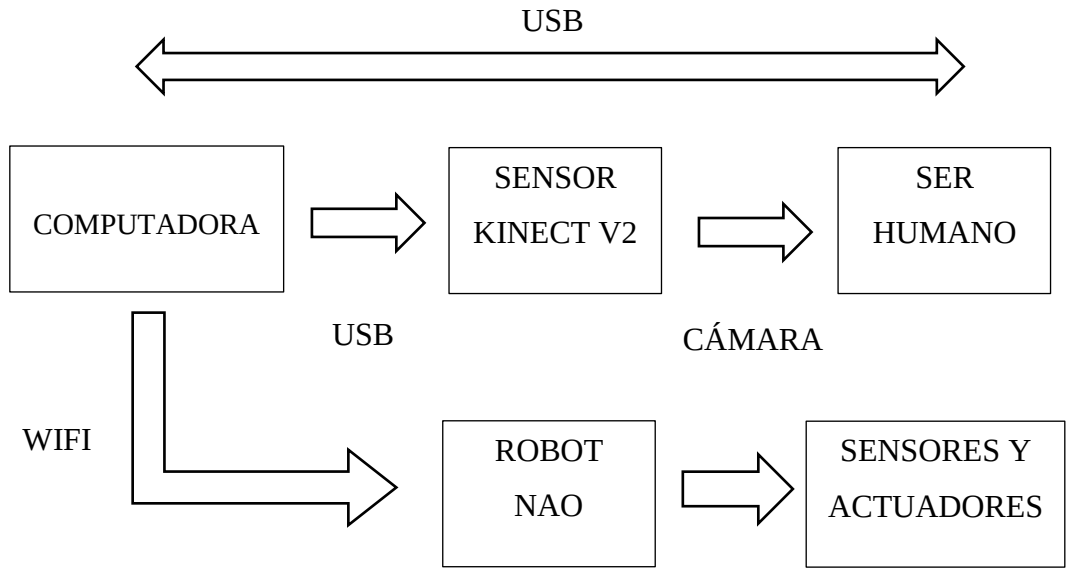
Para su desarrollo en computadoras es necesario incorporar un adaptador que se conecta a un puerto USB 3.0 y el kit de desarrollo de software que proporciona Microsoft (SDK). También se puede adaptar a otras plataformas como Linux gracias herramientas y librerías que han facilitado desarrolladores en todo el mundo.

La herramienta que permite desarrollar aplicaciones para Kinect se denomina SDK (System Development Kit) que se encuentra de manera gratuita en el sitio oficial de Microsoft, para instalarlo es necesario tener sistema operativo Windows 8 o superior, además de Microsoft Visual Estudio para generar el código en lenguaje de programación C#.

CAPÍTULO 3
DISEÑO E IMPLEMENTACIÓN

3.1 Diagrama de bloques del Hardware

Figura 3.1. Hardware del Sistema de control



Partes que intervienen en el sistema de control del robot Nao

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.1 muestra el diagrama de bloques del hardware del sistema de control que está compuesto por la computadora que se encarga de hacer todo el procesamiento de la información, el sensor Kinect V2 que envia la informacion de reconicimiento del esqueleto humano a traves de su cámara y el robot Nao que recoge esos datos y ejecuta la acción segun sea el caso. La comunicación entre el computador y el dispositivo Kinect V2 se realize de forma alámbrica a través del pueto USB, mientras que la comunicación entre la computadora y el robot Nao fué mediante conexión inalambrica utilizando el protocolo Wifi.

3.2 Diagrama de bloques del Software

Figura 3.2. Software del Sistema de control

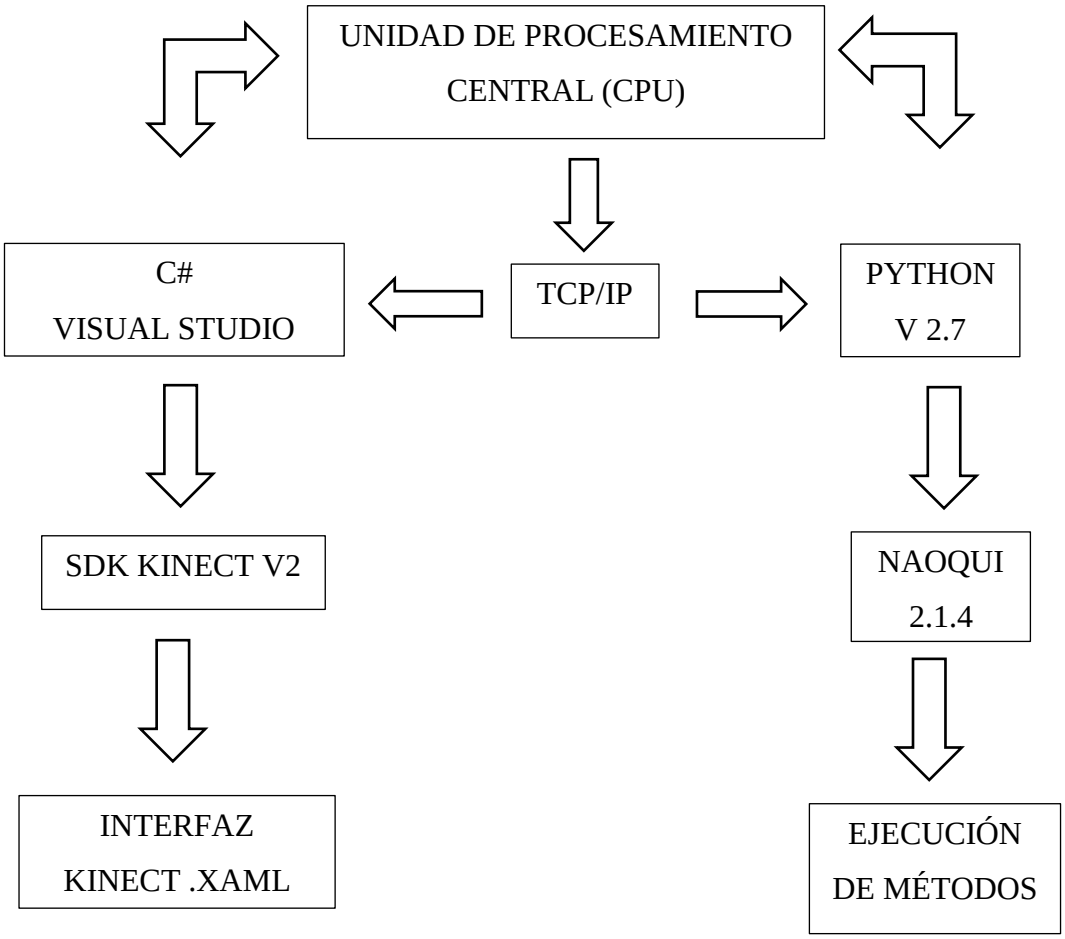


Diagrama de bloques del funcionamiento del Sistema de control

Elaborado por: David Galarza y Christian Llumiquinga

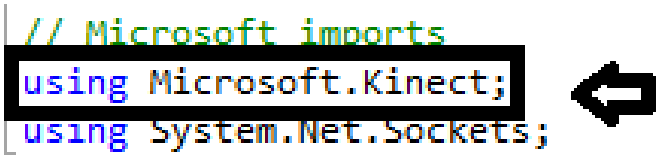
La Figura 3.2 muestra el diagram de bloques del software del Sistema de control donde la computadora es la encargada procesar todos los datos que llegan tanto del Kinect V2 como del robot Nao a través del protocol TCP/IP, es decir que se establece la comunicación dentro del mismo ordenador por puertos locales, ambos lenguajes se ejecutan de forma paralela donde python actúa como servidor y C# como cliente para enviar y recibir información.

3.3 Interfaz para la detección del esqueleto entre Kinect V2 y el humano

Como se mencionó en el capítulo anterior el dispositivo Kinect al ser diseñado por la compañía de Microsoft y ser liberado su código para el desarrollo de aplicaciones de forma libre y gratuita, permite el uso de sus librerías y funciones que vienen incluidas en el Kit de desarrollo integrado (SDK), razón por la cual se escogió el desarrollo del código fuente sobre el sistema operativo Windows 10. La plataforma utilizada fue Visual Studio 2013 ya que permite el desarrollo de código y aplicaciones de forma sencilla utilizando entornos gráficos como herramientas WPF y XAML, en distintos lenguajes de programación mencionando que el proyecto se desarrolló en leguaje C#.

La librería que permite el uso del dispositivo Kinect se denomina Microsoft Kinect la cual debe ser referenciada dentro de nuestro proyecto en visual Studio antes del desarrollo de cualquier algoritmo.

Figura 3.3. Librería de Kinect V2



Código que permite utilizar las librerías de Kinect V2

Elaborado por: David Galarza y Christian Llumiquinga

EL Kit de Desarrollo Integrado de Kinect incorpora ejemplos básicos de cómo utilizar sus librerías para crear aplicaciones tales como: Reconocimiento de audio, captura de audio a través de los micrófonos, reconocimiento de marcos del cuerpo humano, visualización de marcos de color, capturas de imagen, mapeo, entre otros. Razón por la cual gran parte del código fue obtenido de esta herramienta.

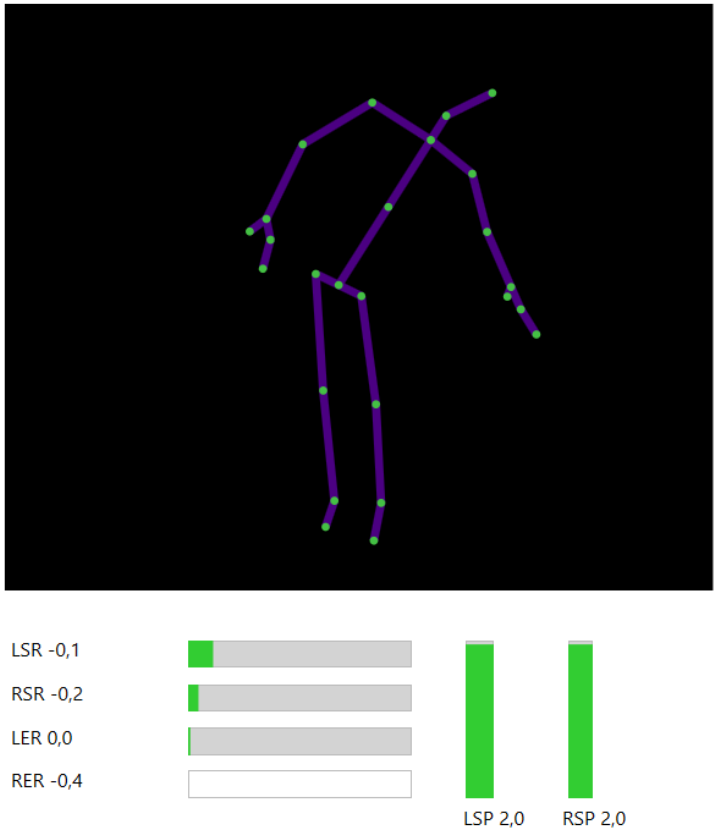
La versión 2 de Kinect permite rastrear hasta 25 articulaciones del esqueleto humano incluyendo los puños y pulgares lo cual representa una parte muy importante en el control del robot Nao. Los pasos más importantes de tomar en cuenta para la detección del esqueleto fueron los siguientes:

- Obtener los datos del cuerpo verificando si son o no nulos.
- Llamar a los métodos correspondientes

- Identificar cada articulación tomando en cuenta que el sensor proporciona posiciones en coordenadas (X, Y, Z), horizontal, vertical y profundidad.
- Acceder a cada articulación una vez conocida su posición.
- Dibujar cada articulación utilizando objetos, líneas, etc.

En la Figura 3.4 se observa la imagen del esqueleto humano generado por el sensor Kinect V2 dentro de una aplicación .XAML

Figura 3.4. Esqueleto humano generado por Kinect V2

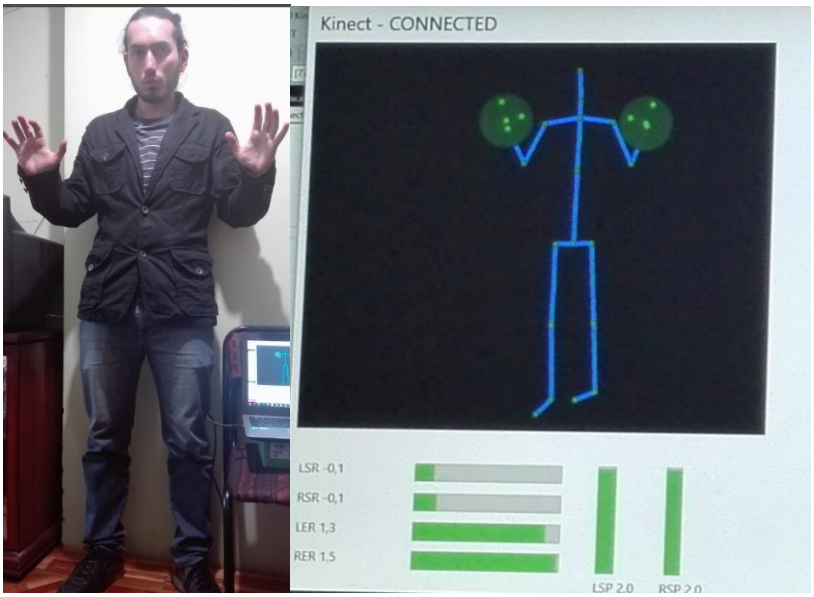


Detalle de las articulaciones y datos proporcionados por Kinect V2

Elaborado por: David Galarza y Christian Llumiquinga

En la Figura 3.5 se muestran la postura de la persona al interactuar con el sensor Kinect V2 y el detalle de las manos cuando se encuentran en posición abiertas se dibujan sobre ellas círculos de color verde y cuando se encuentran en posición cerrada se vuelven de color rojo como se muestra en la Figura 3.6.

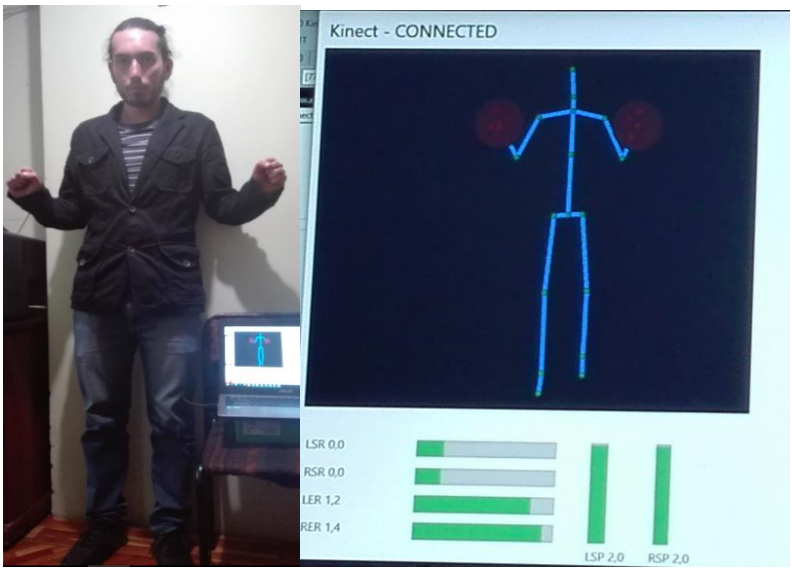
Figura 3.5. Reconocimiento de las manos abiertas



Kinect v2 obtiene los datos de las articulaciones de las manos en posición abiertas

Elaborado por: David Galarza y Christian Llumiquinga

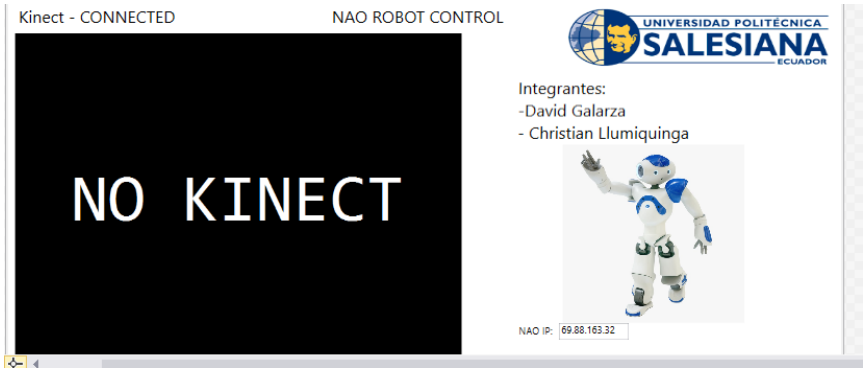
Figura 3.6. Reconocimiento de las manos cerradas



Kinect v2 obtiene los datos de las articulaciones de las manos en posición cerradas

Elaborado por: David Galarza y Christian Llumiquinga

Figura 3.7. Interfaz gráfica para el reconocimiento del esqueleto humano.



Interfaz desarrollada en Visual Studio para el reconocimiento del esqueleto humano

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.7 muestra la interfaz gráfica creada para obtener los datos de las articulaciones del ser humano mediante el uso del sensor Kinect V2.

3.4 Comunicación C# con Python

Para establecer la comunicación entre distintos lenguajes es necesario utilizar un protocolo de comunicación donde los datos que genera el Kinect V2 como resultado de la detección del esqueleto humano, sean procesados y manipulados del sitio remoto en nuestro caso un archivo generado en Python, y este envie las instrucciones al SO del robot Nao. La comunicación se realizó mediante protocolo TCP/IP con interfaz Socket para él envió de datos a través de la red ya que ambos procesos se ejecutarán en paralelo sobre un mismo computador.

Figura 3.8. Comunicación entre Python y C#

Comunicación entre distintos lenguajes sobre un mismo host

Elaborado por: David Galarza y Christian Llumiquinga

Por el lado de Python se importa la librería correspondiente al socket "import socket", que permitirá el envió de datos, luego se crea un puerto socket TCP/IP y se enlaza dicho puerto especificando la dirección local y el número de puerto asociado. Finalmente se crea una línea de comando que permite poner el puerto a la espera de una conexión por parte de la o los clientes.

En C# el proceso es similar, importamos las librerías para crear el protocolo de comunicación,

```
"using System.Net.Sockets;"
```

A continuación, se crea el cliente que va a contener la misma dirección local y número de puerto del servidor:

```
System.Net.Sockets.TcpClientclientSocket = new
System.Net.Sockets.TcpClient();"
```

Se utiliza el método "`clientSocket.GetStream()`" para enviar y recibir datos a través de la red creando un objeto de tipo ASCII.

Para enviar los valores de las articulaciones primero debemos crear un vector que contenga todos los datos en forma de cadena del tipo string, luego creamos un método que reciba esta trama, que a su vez será transformada a bytes para ser enviados finalmente a Python. Si la dirección y puerto en ambos lados son correctas, la comunicación queda finalizada.

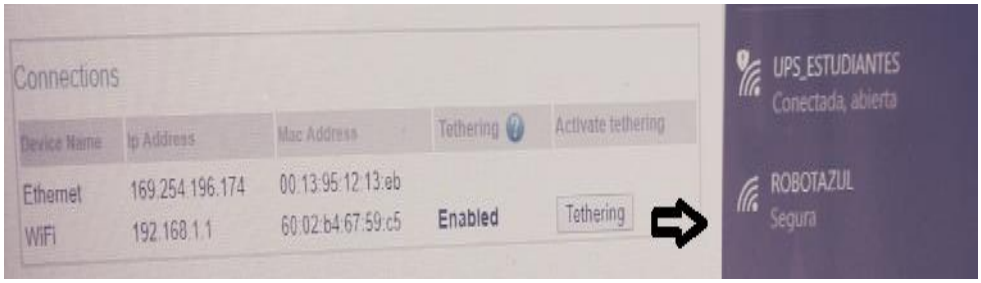
3.5 Análisis y envió de Datos al Robot Nao

Como se mencionó anteriormente la comunicación con el robot se realizó a través de WLAN (Wireless Local Área Network), y utiliza el estándar 802.11g por lo que el método para conectar la computadora con el robot resulta sencillo. Conectamos el robot mediante cable RJ45 directamente al computador, ingresamos la dirección IP por defecto que el robot Nao nos proporciona y dentro de la configuración activamos el

modo inalámbrico para posteriormente conectarnos a la red Wifi que se genera luego de este proceso.

En la Figura 3.9 se muestran en detalle de la conexión a la red inalámbrica.

Figura 3.9. Conexión a la red Inalámbrica del Robot



Conexión entre la computadora y el robot Nao

David Galarza y Christian Llumiquinga

Una vez comprobado que los datos generados por el dispositivo Kinect V2 son recibidos en Python podemos utilizarlos para mover las articulaciones del robot. Hay que tener en cuenta el análisis matemático que se hizo anteriormente y entender cómo funciona la cinemática inversa.

La Tabla 3.1 describe los cálculos matemáticos para los movimientos de las extremidades superiores del Robot tomando como referencia la posición inicial tanto del ser humano como del robot Nao.

Tabla 3.1. Ángulos de posiciones iniciales de las extremidades superiores.

Identificador	**Humano [rad]**	**Nao[rad]**
LSR (Alabeo hombro izquierdo)	-0.30	0
RSR (Alabeo hombro derecho)	-0.30	0
LER (Alabeo codo izquierdo)	-2.70	0.035
RER (Alabeo codo derecho)	-2.70	0.035
LSP (Cabeceo hombro izquierdo)	0.12	1.57
RSP (Cabeceo hombro derecho)	0.12	1.57

Datos de las articulaciones obtenidas en Kinect V2 y el robot Nao en posición normal

Elaborado por: David Galarza y Christian Llumiquinga

$$LSR = (Dato\ Humano + Dato\ Nao)$$
$$LSR = -0.30 + 0$$
$$LSR = -0.30\ [rad]$$

$$RSR = (Dato\ Humano + Dato\ Nao)$$
$$RSR = -0.30 + 0$$
$$RSR = -0.30\ [rad]$$

$$LER = (Dato\ Humano + Dato\ Nao)$$
$$LER = -2.70 + 0.035$$
$$LER = -2.67\ [rad]$$

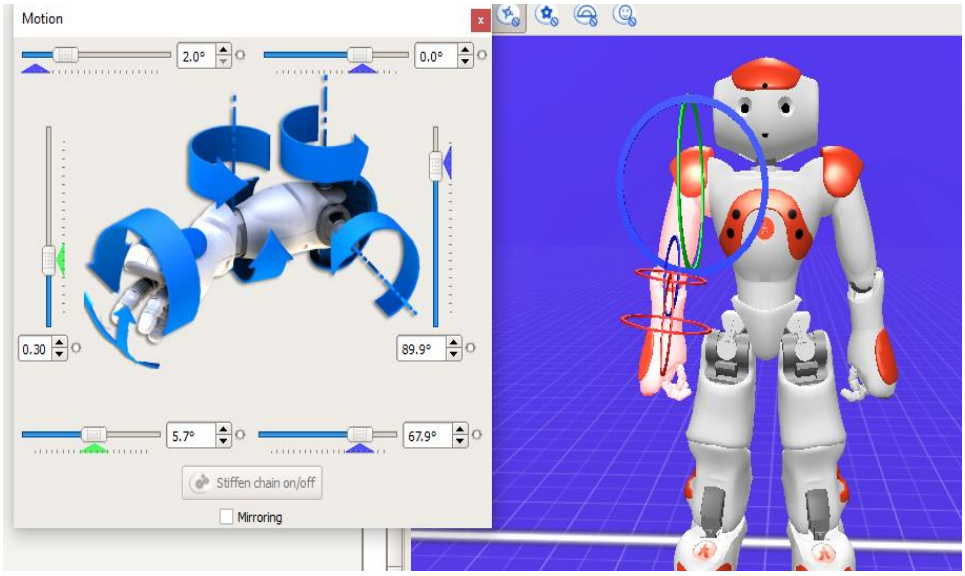
$$RER = (Dato\ Humano + Dato\ Nao)$$
$$RER = -2.70 + 0.035$$
$$RER = -2.67\ [rad]$$

$$LSP = (Dato\ Humano + Dato\ Nao)$$
$$LSP = 0.12 + 1.57$$
$$LSP = 1.69\ [rad]$$

$$RSP = (Dato\ Humano + Dato\ Nao)$$
$$RSP = 0.12 + 1.57$$
$$RSP = 1.69\ [rad]$$

Todos los datos obtenidos de la tabla presentada anteriormente corresponden a la suma de los valores que se observan en las mediciones arrojadas por el sensor Kinect V2 y los valores tomados desde Choregraphe correspondientes al robot. Estos valores pueden variar dependiendo de la postura que sea tomada como referencia.

Figura 3.10. Posición de Referencia del brazo derecho del robot Nao

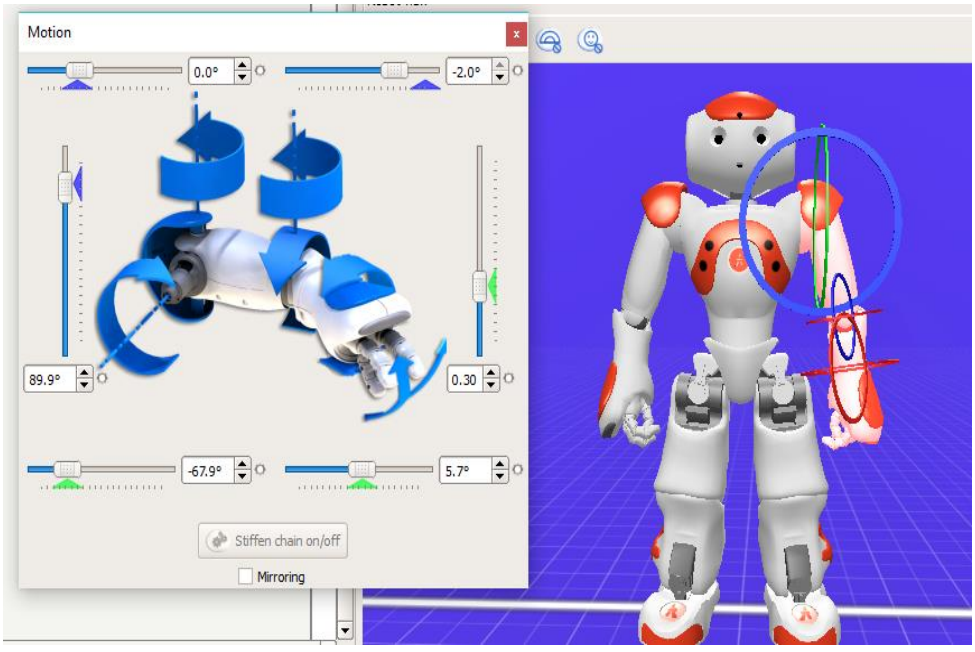


Valores de referencia del brazo izquierdo obtenidos de Choregraphe

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.10 muestra los angulos de referencia del brazo derecho del robot Nao en posición inicial.

Figura 3.11. Posición de Referencia del brazo derecho del Robot Nao



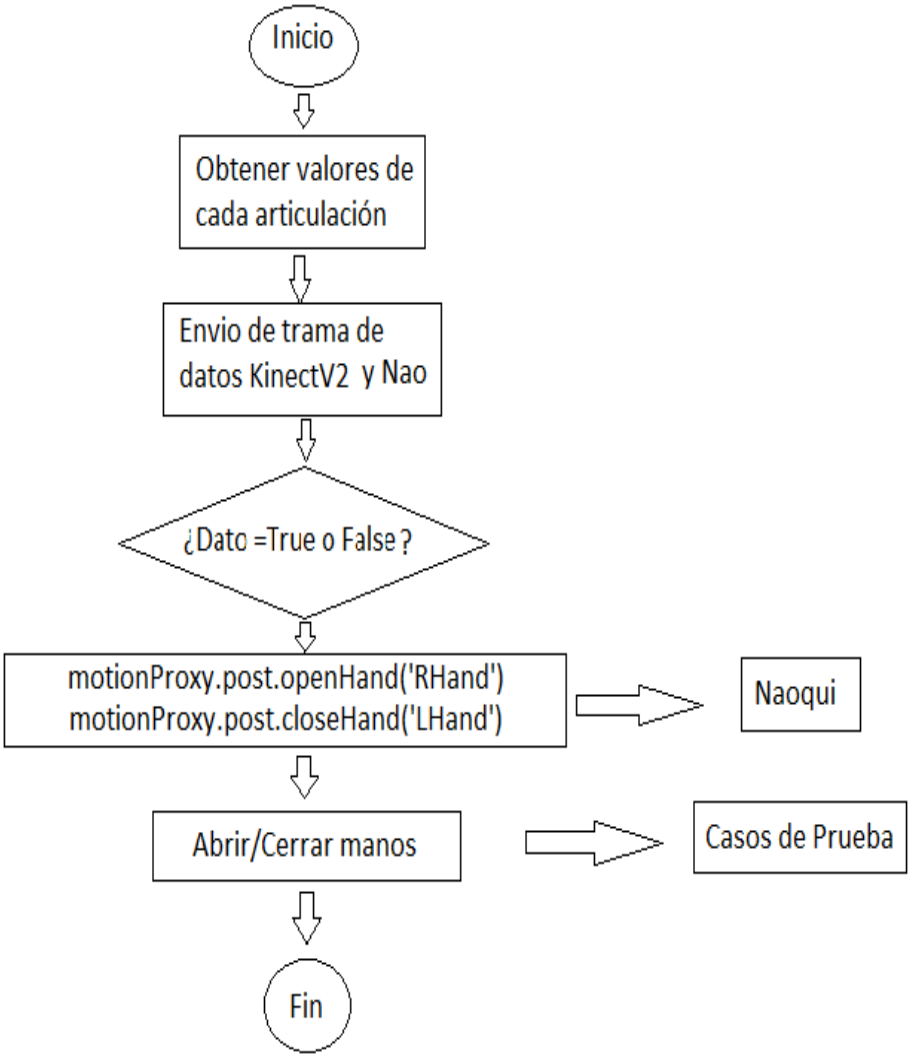
Valores de referencia del brazo derecho obtenidos de Choregraphe

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.11 muestra los angulos de referencia del brazo izquierdo del robot Nao en posición inicial.

La Figura 3.12 muestra el diagrama de flujo para el control de las manos del robot Nao.

Figura 3.12. Proceso de control de movimiento de manos



Apertura y cierre de manos

Elaborado por: David Galarza y Christian Llumiquinga

Para entender claramente el concepto es necesario considerar la programación utilizada para el movimiento de las manos como se muestra en la Figura 3.13.

Figura 3.13. Código para movimiento de manos

```
# CONTROL MANOS
# Izquierda
if (val12[3:7] != str(lastval12)):
                #time.sleep (0.3)

     if (val12[3:7] == 'True'):

               #print "abro"

                 L = True

                 if L == True:
                         motionProxy.post.openHand('LHand')
                         L = False
                         val12[3:7] == 'True'

     if (val12[3:8] == 'False'):

               #print "cierro"


                  L = False


                  if L == False:
                          motionProxy.post.closeHand('LHand')
                          L = True
                          val12[3:8] == 'False'
                  time.sleep (0.01)

lastval12 = val12[3:7]


# Derecha
if (val11[3:7] != str(lastval11)):
                #time.sleep (0.3)

                if (val11[3:7] == 'True'):

                            #print "abro"
```

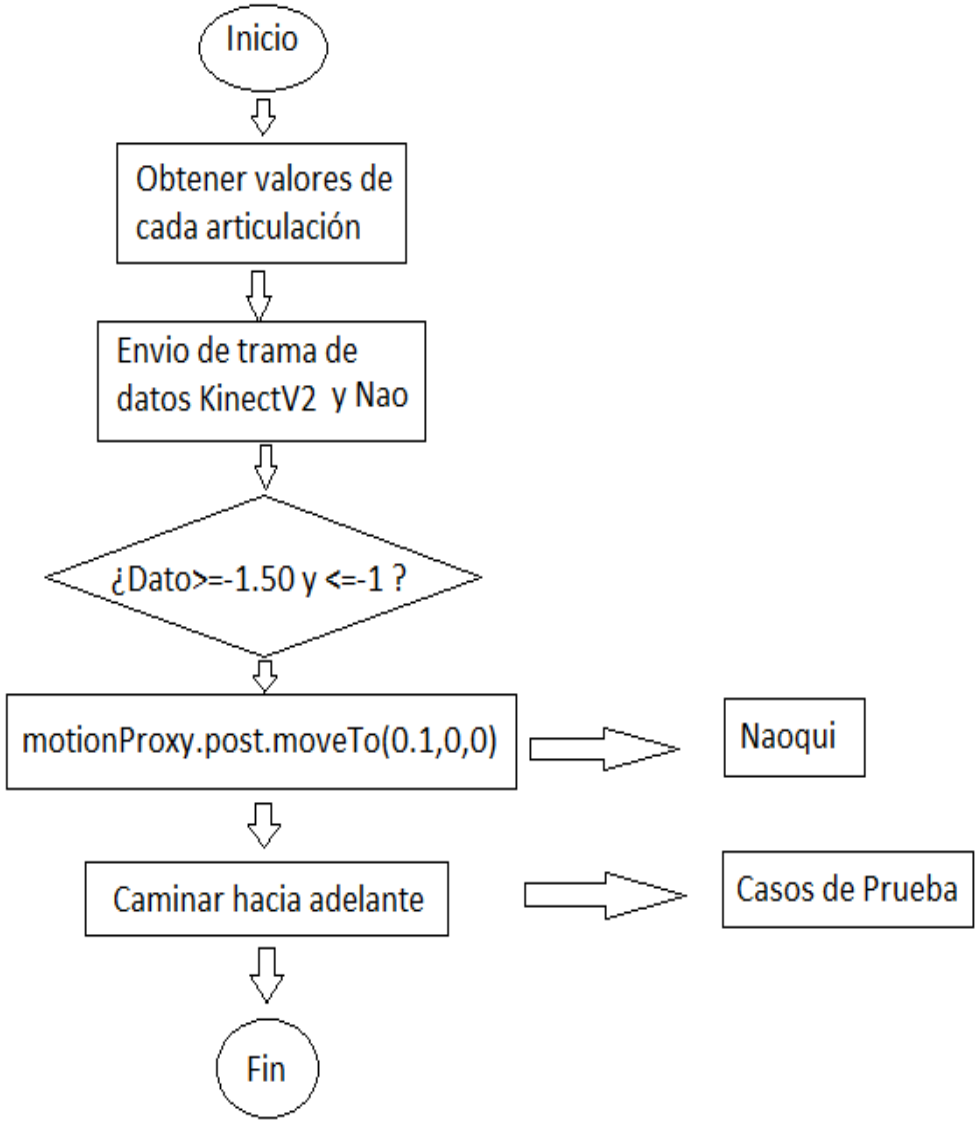
Código para el movimiento de las manos del robot Nao

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.13 muestra el control de la mano izquierda. El código compara si el dato que llega es verdadero o falso, si la condición es verdadera ejecuta el método que permite al robot abrir su mano caso contrario el robot cierra su mano.

La Figura 3.14 muestra el proceso para el control de movimiento del robot Nao hacia adelante.

Figura 3.14. Proceso de control de movimientos hacia adelante



Desplazamiento hacia adelante

Elaborado por: David Galarza y Christian Llumiquinga

Por ejemplo, si queremos movernos hacia adelante el código para realizar esta acción se observa en la Figura 3.15.

Figura 3.15. Código para desplazamiento hacia Adelante

```
#MOVIMIENTO EXTREMIDADES INFERIORES
motionProxy.post.setWalkArmsEnabled(False, False)

if val7[0:3] == 'RHP': #Right Hip pitch

        valh=val7[4:10]
        valh= float(valh)
        valh=(round (valh,2))

        if (valh >=-1.50 and valh<=-1.00):

            motionProxy.post.moveTo(0.1,0,0)#adelante
            time.sleep (0.01)
```

Desplazamiento del robot Nao

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.13 muestra el código para que el robot Nao camine hacia adelante. Se compara si existe un dato con el identificador 'RHP', si existe dicho dato ingresa a la condición, se transforma el valor a tipo flotante con dos decimales, seguido de otra condición que establece en que rango de valores debe estar posicionado la pierna para que el robot la reconozca y ejecute la acción con su respectivo método.

Tabla 3.2. Rango de apertura de las extremidades inferiors.

Coordenada	Juntura	Pierna Derecha [Rad]	Pierna Izquierda [Rad]
Cabeceo	Posición Normal	[0- (- 0.90)]	[0-(-0.90)]
	Levantada	[-1- (-1.5)]	[-1- (-1.5)]
Alabeo	Posición Normal	[0-2]	[0-2]
	Levantada	[3-4]	[3-4]

Valores medidos por el Kinect V2 de las piernas en posición normal y levantada

Elaborado por: David Galarza y Christian Llumiquinga

La Tabla 3.2 corresponde a los intervalos de las posiciones de las extremidades inferiores realizadas por el movimiento del ser Humano detectado por el dispositivo Kinect V2.

Para el caso de las extremidades inferiores se utilizó el método

- motionProxy.post.moveTo(X, Y, Theta)

Donde:
X: Corresponde al valor en metros (m) del desplazamiento hacia adelante y negativos hacia atrás.
Y: Corresponde al valor en metros (m) de los desplazamientos laterales (izquierda/derecha).
Theta: Corresponde a la rotación en su propio eje (sentido horario o antihorario).

También se consideró el movimiento de la muñeca para que el robot sea capaz de tomar algún objeto que se encuentre en una superficie o que requiera un mejor posicionamiento. Para ello que utilizó el reconocimiento de la cabeza por parte del Kinect V2 y con los datos obtenidos al inclinar la cabeza se generó dicho movimiento.

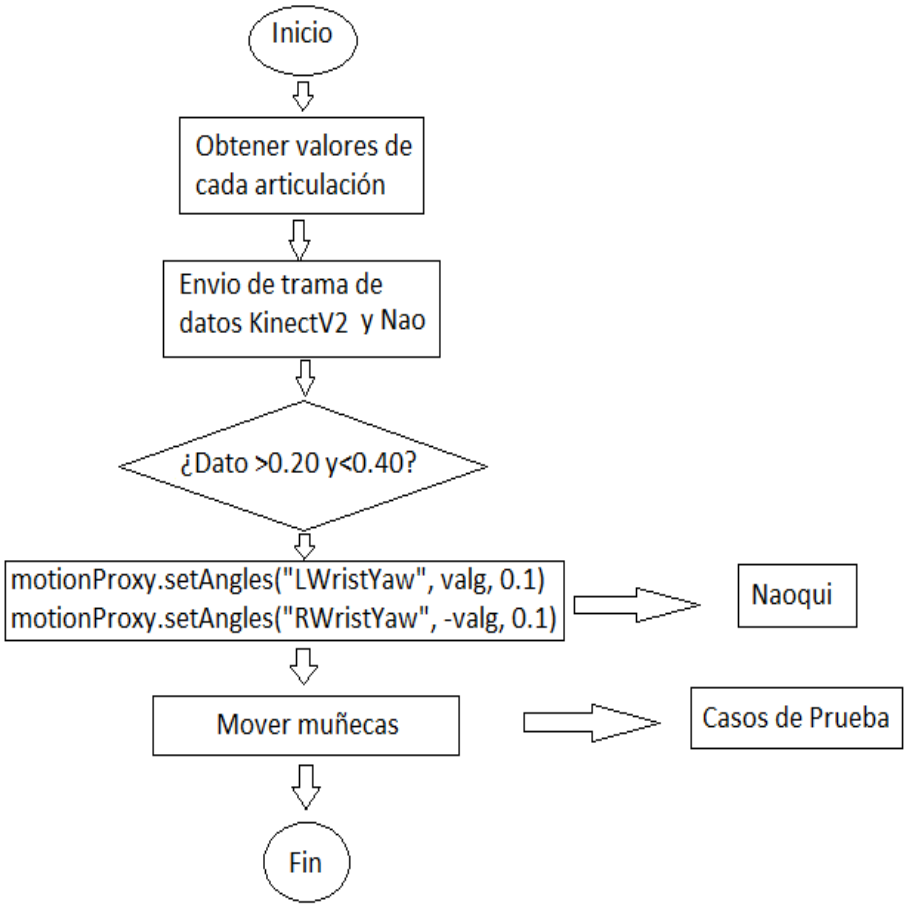
Tabla 3.3. Valores obtenidos por Kinect V2 al mover la cabeza

Posición	**Juntura**	**Humano [Rad]**	**Nao [Rad]**
Normal	Cabeza (Alabeo)	[0.01 – 0.04]	0.098
Derecha o Izquierda		[0.10 – 0.40]	1.31

Valores que resultan del movimiento de la cabeza con inclinación hacia la derecha e izquierda
Elaborado por: David Galarza y Christian Llumiquinga

El Diagrama de la Figura 3.16 muestra el proceso para mover las muñecas del robot Nao mediante los datos enviados por el movimiento de la cabeza.

Figura 3.16. Proceso de control de movimiento de muñecas



Giro de muñecas

Elaborado por: David Galarza y Christian Llumiquinga

Por ejemplo, si queremos mover la muñeca el código para realizar esta acción se observa en la Figura 3.17.

Figura 3.17. Código para la rotación de las muñecas

```
)NTROL MUÑECA


val6[0:3] == 'HRL': #Head Roll (mov_muñeca)

      valg=val6[4:10]
      valg= float(valg)
      valg=(round (valg,2))

      if (valg >0.20 and valg<=0.40):

          valg= 1.21+valg
          motionProxy.setAngles("LWristYaw", valg, 0.1)
          motionProxy.setAngles("RWristYaw", -valg, 0.1)

      if (valg >=0 and valg<=0.20):

          valg= 0.35
          motionProxy.setAngles("LWristYaw", valg, 0.1)
          motionProxy.setAngles("RWristYaw", -valg, 0.1)
```

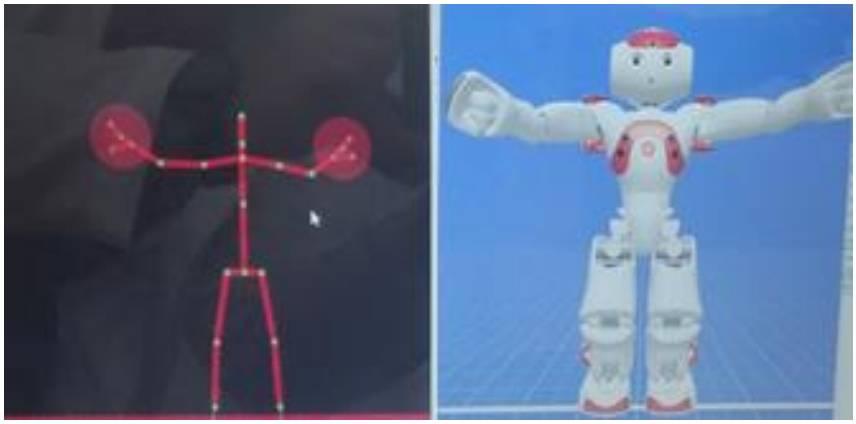
Código para la rotación de las muñecas del robot Nao.

Elaborado por: David Galarza y Christian Llumiquinga

3.6 Simulación de movimientos

Para la simulación y replica de movimientos se utilizó la herramienta Choregraphe que permite visualizar un robot virtual en conjunto con el programa realizado en visual estudio en lenguaje C# y el sensor Kinect V2.

Figura 3.18. Teleoperación del robot Virtual

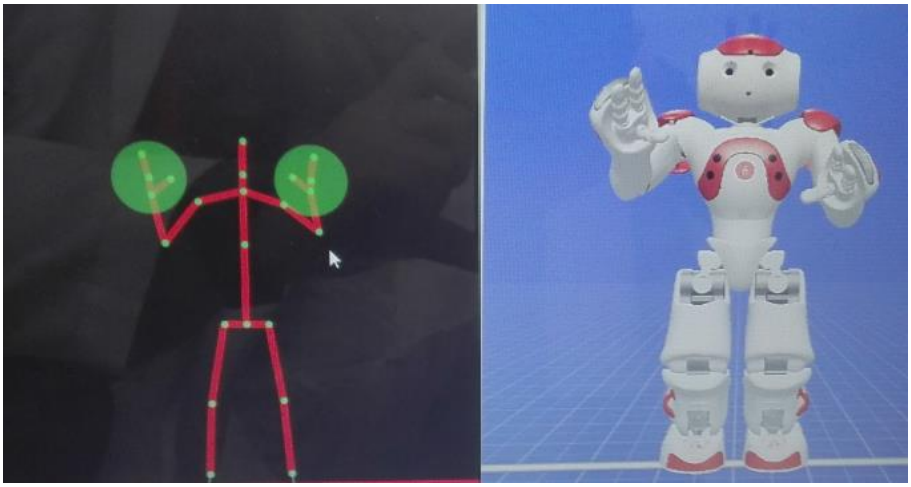


Replica de movimientos con las manos cerradas

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.18 muestra la réplica de movimiento del robot Nao con los brazos abiertos horizontalmente y con las manos cerradas, en la interfaz del Kinect las marcas se identifican con círculos de color rojo. Mientras que en la Figura 3.19 las manos están abiertas por lo que las marcas se muestran con círculos de color verde.

Figura 3.19. Teleoperación del robot virtual



Replica de movimientos con las manos abiertas
Elaborado por: David Galarza y Christian Llumiquinga

En la Figura 3.20 se puede observar la interacción entre la persona que está parada en frente del Kinect, la pantalla de visualización del esqueleto y la herramienta de Choregraphe que permite simular el robot Nao.

Las réplicas del robot Nao resultan similares a las del ser humano con lo que se comprueba el buen funcionamiento al momento de implementar este sistema en un entorno real. El control de movimiento de las extremidades inferiores del robot Nao también pueden ser simuladas, pero a simple vista el desplazamiento no se puede identificar en su totalidad ya que el plano de fondo no está claramente definido en la pantalla de simulación de Choregraphe.

Figura 3.20. Prueba de simulación 1

Réplicas de movimientos en distintas posiciones

Elaborado por: David Galarza y Christian Llumiquinga

Figura 3.21. Prueba de simulación 2

Réplicas de movimientos en distintas posiciones

Elaborado por: David Galarza y Christian Llumiquinga

3.7 Descripción de la arquitectura

En la Figura 3.22 se presenta un diagrama que indica cómo se diseñó el sistema de control de teleoperación del robot Nao en la ejecución de tareas en un determinado entorno.

Figura 3.22. Esquema del sistema de control

Comunicación y control desde el sitio del operador hacia el sitio remoto

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.22 muestra dos partes importantes en nuestro sistema, el lado operador que consta del espacio donde la persona va a realizar el control y replica de movimientos del robot mediante el uso del sensor Kinect V2. Este a su vez está conectado a una laptop la cual proyecta imágenes de lo que está observando el robot a través de sus cámaras incorporadas, y el lado remoto corresponde al entorno donde el robot Nao realizará las tareas de maniobra y transporte.

Una vez que se ejecutan los programas inicia el proceso de reconocimiento a través del sensor Kinect V2, si la persona es detectada a través de las cámaras inmediatamente realiza el envío de datos a través de la red, y el robot replica los movimientos de acuerdo con la posición de la articulación; caso contrario no envía ninguna trama.

3.8 Implementación del entorno

Para la creación del entorno se tomaron en cuenta las características físicas del robot Nao que permiten realizar las tareas de maniobra y transporte de forma correcta.

En primer lugar, se diseñó un marco rectangular que simula una entrada por la cual el robot ingresara de forma autónoma, a continuación, se colocaron distintos obstáculos sobre la piso que tendrán que ser evadidos por el robot Nao para luego llegar a un panel con un final de carrera que al ser presionado activará un foco indicando la realización de la tarea, luego el robot se desplaza hasta llegar a un aro que cuelga de la mesa, el robot sujetara dicho anillo y lo depositará en una caja para finalmente terminar con su trabajo.

Materiales utilizados:

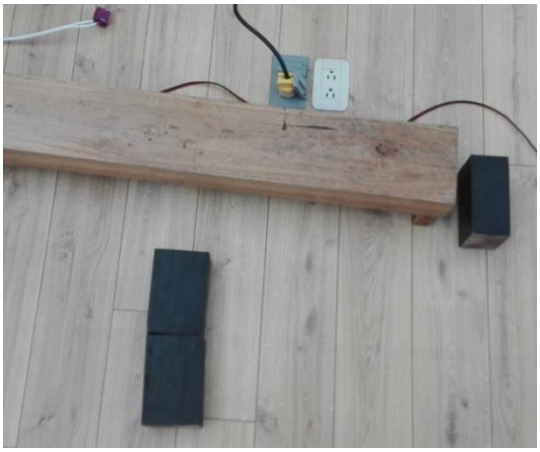
- Tabla Triplex
- Piezas de madera
- Final de Carrera
- Cinta aislante
- Cinta doble fase
- Caja de cartón

Figura 3.23. Puerta de entrada

Puerta de entrada de aproximadamente 40 cm

Elaborado por: David Galarza y Christian Llumiquinga

Figura 3.24. Bloq1ues de madera



Bloque de madera que representan los obstáculos

Elaborado por: David Galarza y Christian Llumiquinga

En la Figura 3.24 se observa bloques de madera de tal forma que el robot Nao al encontrarse con dicho obstáculo sea capaz de evadirlo moviéndose hacia un costado

para luego continuar con el recorrido hasta llegar al tablero donde presionará un interruptor que encenderá el foco.

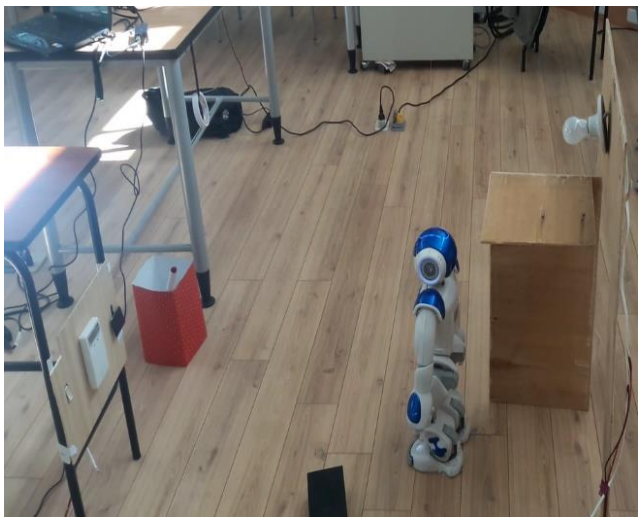
Figura 3.25. Tablero de control con final de Carrera

Panel con final de carrera que enciende un foco

Elaborado por: David Galarza y Christian Llumiquinga

Los demás elementos que se observan en la Figura 3.25 fueron inicialmente incluidos en nuestro trabajo, pero luego se descartaron ya que se consideró que el robot Nao no estaba en la capacidad de realizarlas.

Figura 3.26. Entorno rea de prueba



Visualización del entorno real

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 3.26 muestra el entorno donde se realizó las pruebas pertinentes.

CAPÍTULO 4

PRUEBAS Y RESULTADOS

4.1 Pruebas de réplica estáticas

En el presente capitulo se detalla las pruebas realizadas en la teleoperación del robot nao con el sensor Kinect v2. Las pruebas constan de dos partes para verificar la réplica de movimientos del robot nao por medio del sensor Kinect v2. Se realizó una serie de ejercicios para verificar el control de los movimientos del robot Nao por medio del sensor Kinect v2 estas pruebas se las hizo confirmando que el robot Nao replique los movimientos de las extremidades superiores, así como también el desplazamiento en un entorno abierto ejecutadas por parte del teleoperador.

Los resultados obtenidos de una serie de ejercicios de rutinas se registraron en la tabla 4.1.

Tabla 4.1. Tabla de resultados de pruebas

OBJETIVOS	Nº DE PRUEBAS	PRUEBAS SATISFACTORIAS		PRUEBAS ERRADAS	
		Nº PRUEBAS	%	Nº PRUEBAS	%
Movimiento del brazo derecho	10	9	90	1	10
Movimiento del antebrazo derecho	10	10	100	0	0
Sujeción con los dedos de la mano derecha	10	9	90	1	10
Giro de la muñeca derecha	10	9	90	1	10
Movimiento del brazo izquierdo	10	9	90	1	10
Movimiento del antebrazo izquierdo	10	9	90	1	10
Sujeción con los dedos de la mano izquierda	10	9	90	1	10
Giro de la muñeca izquierda	10	9	90	1	10
Desplazamiento hacia adelante	10	9	90	1	10
Desplazamiento hacia atrás	10	9	90	1	10
Desplazamiento hacia la izquierda	10	9	90	1	10
Desplazamiento hacia la derecha	10	9	90	1	10
Desplazamiento en un entorno para la ejecución de tareas específicas	5	4	80	1	20
PROMEDIO DE RESULTADOS DE LAS PRUEBAS EJECUTADAS			90		

Porcentajes de resultados de pruebas

Elaborado por: David Galarza y Christian Llumiquinga

$$Porcentaje\ de\ Pruebas = \frac{Numero\ de\ pruebas\ correctas}{Numero\ de\ pruebas\ totales} x100\% \quad \text{EC (4.1)}$$

$$Promedio\ total = \sum \left(\frac{Porcentaje\ total\ de\ pruebas}{Numero\ de\ objetivos\ realizados} \right) \quad \text{EC (4.2)}$$

$$\%Error = \sum \left(\frac{Porcentaje\ de\ pruebas\ Erradas}{Numero\ de\ objetivos\ realizados} \right) \quad \text{EC (4.3)}$$

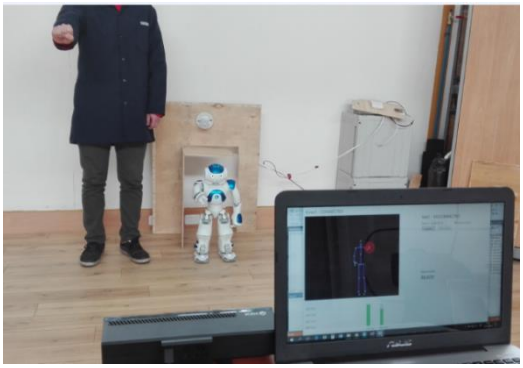
En las Figuras 4.1, 4.2 y 4.3 se observa la réplica de movimiento del brazo derecho, observando el correcto funcionamiento de la extremidad salvo el caso en la ejecución Rshoulderroll ya que por motivos de hardware del robot Nao tiene averías en la articulación y no puede mover a esa posición.

Figura 4.1. Movimiento del brazo derecho

Replica de movimiento del brazo derecho

Elaborado por: David Galarza y Christian Llumiquinga

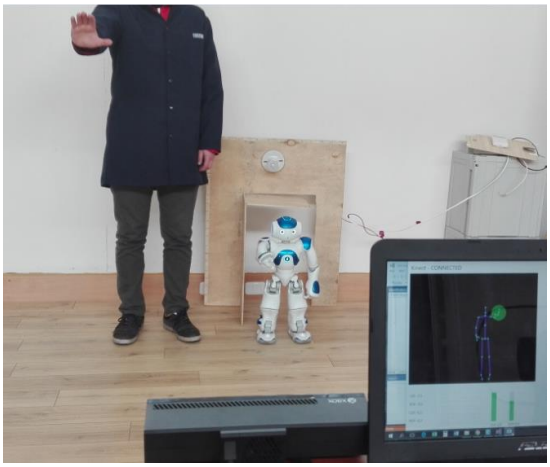
Figura 4.2. Movimiento de los dedos de la mano derecha



Replica de movimiento de la mano derecha

Elaborado por: David Galarza y Christian Llumiquinga

Figura 4.3. Movimiento de la muñeca de la mano derecha



Giro de la muñeca de la mano derecha.

Elaborado por: David Galarza y Christian Llumiquinga

En las Figura 4.4 se observa la réplica de movimiento del brazo izquierdo, observando el correcto funcionamiento de la extremidad salvo el caso en la ejecución guiñada codo izquierdo ya que por motivos de hardware del robot Nao tiene averías en la articulación y no puede mover a esa posición

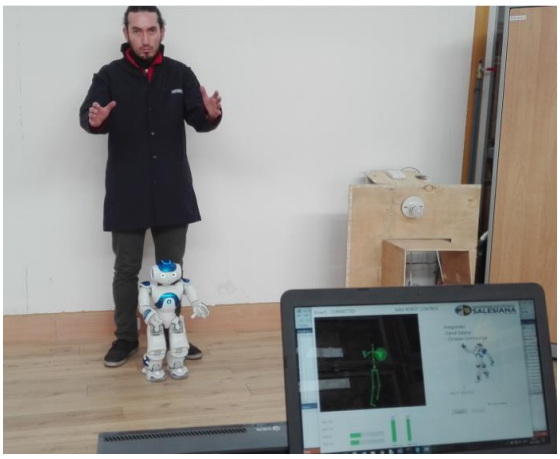
Figura 4.4. Movimiento del brazo izquierdo

Replica de movimiento del brazo izquierdo.

Elaborado por: David Galarza y Christian Llumiquinga

En la figura 4.5 se observa la réplica de movimiento para los dos brazos.

Figura 4.5. Movimiento de los brazos

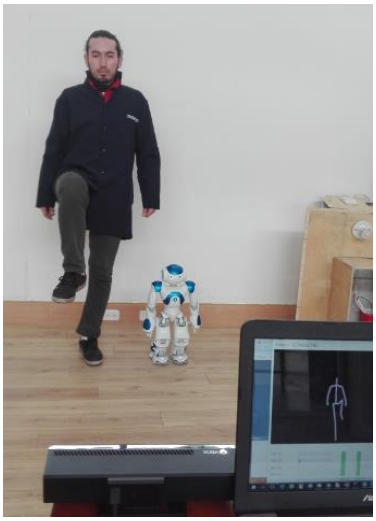


Replica de movimiento de los brazos.

Elaborado por: David Galarza y Christian Llumiquinga

En la Figura 4.6 se observa la postura del teleoperador debe ejecutar para que el Robot Nao se desplace cuatro pasos hacia adelante, la postura que debe reconocer el sensor Kinect V2 es subir la rodilla derecha formando un ángulo de 90º con respecto a la cintura.

Figura 4.6. Postura para desplazamiento hacia adelante

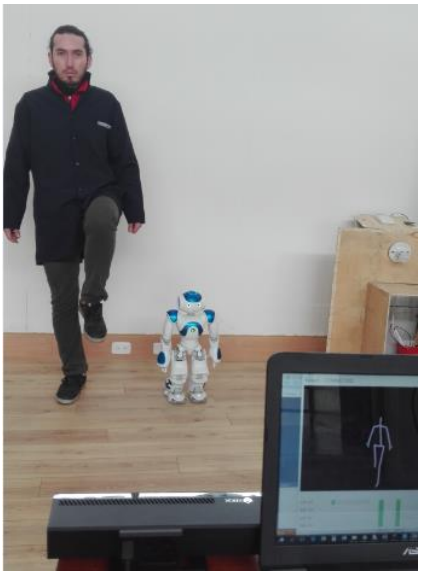


Postura para desplazamiento 0.1m hacia adelante.

Elaborado por: David Galarza y Christian Llumiquinga

En la Figura 4.7 se observa la postura del teleoperador debe ejecutar para que el Robot Nao se desplace cuatro pasos hacia atrás, la postura que debe reconocer el sensor Kinect V2 es subir la rodilla izquierda formando un ángulo de 90º con respecto a la cintura.

Figura 4.7. Postura para desplazamiento hacia atrás

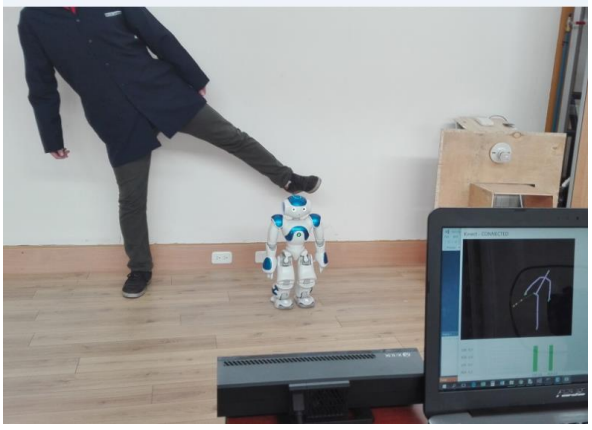


Postura para desplazamiento 0.1 m hacia atrás.

Elaborado por: David Galarza y Christian Llumiquinga

En la Figura 4.8 se observa la postura del teleoperador debe ejecutar para que el Robot Nao se desplace cuatro pasos que corresponde a 0.1m hacia la izquierda.

Figura 4.8. Postura para desplazamiento hacia la izquierda

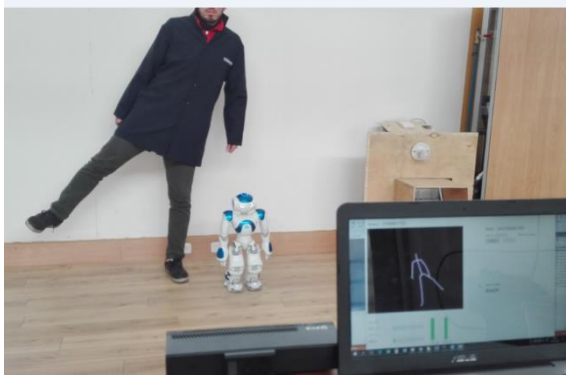


Postura para desplazamiento de 0.1m hacia la izquierda.

Elaborado por: David Galarza y Christian Llumiquinga

En la Figura 4.9 se observa la postura del teleoperador debe ejecutar para que el Robot Nao se desplace cuatro pasos que corresponde 0.1m hacia la izquierda.

Figura 4.9. Postura para desplazamiento hacia la derecha



Postura para desplazamiento de 0.1m hacia la derecha

Elaborado por: David Galarza y Christian Llumiquinga

En la Figura 4.10 se observa la sujeción de un objeto con la mano derecha, el teleoperador tiene el control tanto de giro de la muñeca como de los dedos del robot tomando en cuenta las características del objeto en peso y tamaño.

Figura 4.10. Sujeción de objeto con la mano derecha

Réplica de manipulación de un objeto

Elaborado por: David Galarza y Christian Llumiquinga

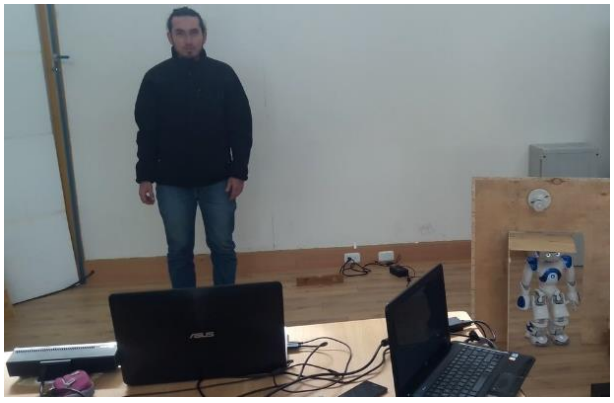
4.2 Pruebas de funcionamiento en el entorno

Para las pruebas de funcionamiento en el entorno se tomó en consideración dos aspectos importantes:

- Actividades que requieran la manipulación de objetos utilizando el movimiento de extremidades superiores del robot, incluyendo las manos.
- Actividades que requieran el desplazamiento del robot a través de una superficie con obstáculos.

1.- Prueba de ingreso del Robot Nao por acceso con medias reducidas.

Figura 4.11. Interacción entre humano y robot Nao



Prueba de ingreso por la puerta de acceso

Elaborado por: David Galarza y Christian Llumiquinga

La Figura 4.11 muestra el ingreso del robot Nao a través de un orificio acorde a las medidas del robot Nao. Uno de los problemas presentados en esta prueba fue con respecto a la superficie del piso ya que al estar hecha de madera no hay la suficiente fricción con la superficie de los pies del robot por lo que tiende a desviar su trayectoria conforme avanza en su recorrido. También es importante mencionar que la ubicación de brazos al momento del desplazamiento debe estar posicionadas correctamente, caso contrario el robot tiende a cambiar su centro de masa lo cual provocaría pérdida de equilibrio y podría caerse en el trayecto.

2.- Prueba de desplazmiento por obstáculos.

Figura 4.12. Nao caminando a través de obstáculos

Desplazamiento del robot Nao a través de obstáculos

Elaborado por: David Galarza y Christian Llumiquinga

Para esta actividad es necesario tener un gran control de las extremidades inferiores del cuerpo. El método utilizado para que el robot se desplace ya sea hacia la izquierda o bien hacia la derecha fue mediante movimiento de las piernas de forma horizontal como se observa en la Figura 4.13.

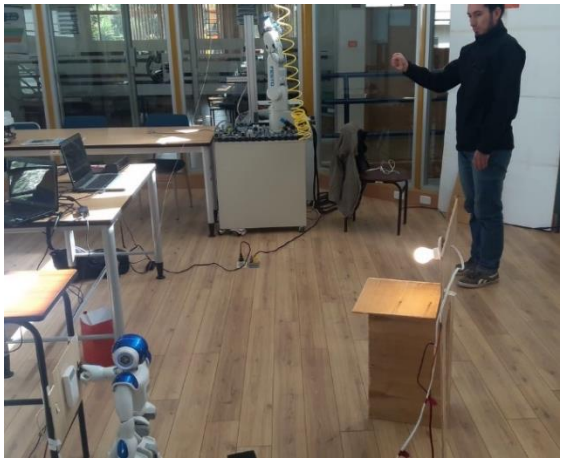
Figura 4.13. Desplazamiento lateral del robot Nao

Desplazamiento del robot Nao a través de obstáculos

Elaborado por: David Galarza y Christian Llumiquinga

3.- Prueba de pulsación de interruptor para el encendido de foco.

Figura 4.14. Encendido de foco



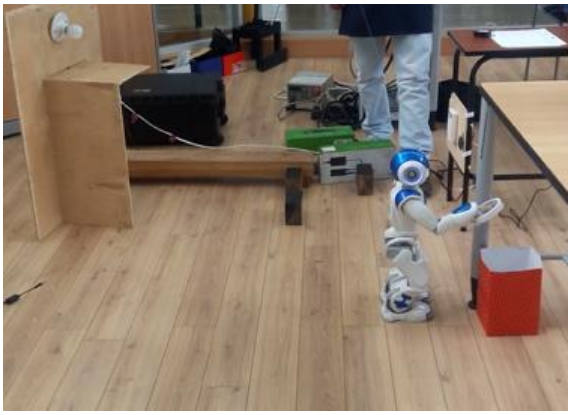
Encendido de foco mediante réplica de movimiento

Elaborado por: David Galarza y Christian Llumiquinga

Para que la tarea sea realizada con éxito se consideró que la posición de la mano se encuentre cerrada al momento de presionar el final de carrera.

4.- Prueba de sujeción de objeto y deposito en caja.

Figura 4.15. Traslado y manipulación de objeto



Traslado y manipulación de objeto

Elaborado por: David Galarza y Christian Llumiquinga

Para esta tarea es importante tener control sobre las muñecas del robot a través del movimiento de la cabeza, de tal manera que estas se posicionen horizontalmente, pueda sujetar el objeto con las manos y luego transportarlo hacia la caja.

CONCLUSIONES

Se teleoperó al robot Nao por medio de los sensores del dispositivo Kinect v2 logrando replicar los movimientos emitidos por el teleoperador realizando una serie de movimientos de las extremidades con un índice de efectividad de un 90%, y el 10% de error contempla fallas de hardware del robot Nao, correspondiente a la articulación del hombro del brazo derecho y a la articulación del codo del brazo izquierdo.

Se realizó la comunicación entre el sensor Kinect v2 y el software de desarrollo para la detección del esqueleto usando como herramienta Visual Studio y lenguaje de programación C# en .NET, el sensor Kinect v2 es propiedad de Microsoft logra una buena integración con la computadora sistema operativo Windows, también incluye aplicaciones basadas en lenguaje de programación nativos como C#, C++, que facilitan el desarrollo de nuevas tecnologías como en el caso de la robótica.

Se simuló el entorno teniendo en cuenta las limitaciones de hardware del robot Nao obteniendo un índice de efectividad del 80% en las pruebas de réplica dinámica como se observa en la Tabla 4.1. El 20% de error corresponde al hardware del robot, en parte se debe a las articulaciones dañadas que limitan la replica exacta de los movimientos y al ìndice de rozamiento que existe entre la base de los pies del robot y el piso que hace que se desvie algunos grados al caminar en linea recta, solventando en gran medida al colocar papel antideslizante en la base de los pies del robot Nao.

Se concluye que el sensor Kinect v2 además de ser un controlador de juegos se le puede dar un uso investigativo y desarrollar nuevas aplicaciones. El sensor Kinect v2 tiene una resolución 1920 x 1080 dándonos una mayor precisión en detectar cada punto del cuerpo humano incluyendo los dedos de las manos, además tiene un puerto USB 3.0 que mejora la velocidad de transmisión obteniendo mejor respuesta al momento de realizar las réplicas en tiempo real.

Se comprobó que la mejor técnica para intercambiar flujo de datos entre la aplicación realizada en Visual estudio y Phyton fue mediante el protocolo TCP/IP utilizando puerto Socket ya que su implementación requiere de una arquitectura cliente servidor para que la información sea transmitida en toda la red una vez que se establece la comunicación.

RECOMENDACIONES

Es importante considerar que la posiciòn del sensor Kinect V2 con respecto a la persona que va a controlar el robot Nao debe estar a una distancia sugerida de 1.5m y el Kinect V2 a una altura de 1m para que la detección tanto de las extremidades superiores como inferiores sean correctas, de lo contrario los datos no serán reconocidos adecuadamente y en algunos casos se pierde la comunicación con el robot.

Se recomienda que los objetos que va a manipular el robot Nao estén dentro de los parámetros de peso y dimensiones como se indica en los datos del fabricante, para evitar que pierda el equilibrio al momento de moverse o transportar el objeto de un lugar a otro.

La distancia entre el Robot y el computador no debe exceder los 12 metros de distancia a la redonda ya que al utiliza una conexión inalámbrica (WLAN), y tomando en cuenta el entorno en general puede afectar la comunicación.

La superficie donde se moviliza el robot no deber ser del todo lisa ni tampoco rugosa ya que esto afecta su trayectoria, también es preciso mencionar que la superficie debe ser plana y sin inclinación.

REFERENCIAS

Ballesteros, S. A. (Octubre de 2012). *Sistema de teleoperación mediante interfaz natural de usuario.* Obtenido de http://e-archivo.uc3m.es/bitstream/handle/10016/16682/PFC_Santiago_Alfaro_Ballesteros.pdf?sequence=1

Carmona, M. P. (Miercoles de Marzo de 2016). *El Megáfono*. Obtenido de https://www.megafonofcom.es/ciencia-tecnologia/el-nuevo-robot-atlas

Duque, E. (03 de Febrero de 2015). *Que es Kinect*. Recuperado el 3 de Mayo de 2016, de https://edwinnui.wordpress.com/2015/02/03/qu-es-el-microsoft-kinect/

EcuRed. (13 de 12 de 2017). Obtenido de https://www.ecured.cu/Lenguaje_de_Programaci%C3%B3n_C_Sharp

Foundation, O. S. (04 de Abril de 2014). *ROS.org*. Recuperado el 17 de Junio de 2016, de http://wiki.ros.org/es

Galarza David, L. C. (2017). *Interfaz Choregraphe.* Quito.

Gold Binary Robot. (2016). Recuperado el 12 de Mayo de 2016, de http://www.gold-binary-robot.com/es/robot-de-opciones-binarias

Honda. (2015). *Honda The Power of Dreams*. Recuperado el 8 de Mayo de 2016, de https://www.honda.mx/asimo/

Intorobotics. (05 de Mayo de 2014). *Intorobotics*. Recuperado el 13 de Mayo de 2016, de https://www.intorobotics.com/ros-tutorials-start-working-arduino-raspberry-pi/

Ishiguro, H. (2016). *ATR Home*. Obtenido de http://www.geminoid.jp/en/robots.html

Kofinas, N. (Julio de 2012). *Forward and Inverse Kinematics for the NAO Humanoid Robot.* Obtenido de https://www.cs.umd.edu/~nkofinas/Projects/KofinasThesis.pdf

Kowalczyk, T. (Septiembre de 2011). *Kinect SDK, Natural User Interface API*. Recuperado el 7 de Mayo de 2016, de https://msdn.microsoft.com/pl-pl/library/kinect-sdk--natural-user-interface-api.aspx

Kumar, S. S. (2010). *Introducción a la robótica.* Ciudad de Mexico: Tata Mcraw-Hill Education. Recuperado el 12 de Mayo de 2016, de https://books.google.com.ec/books?id=T_N94WFPLdIC&pg=PA122&dq=tipos+de+paracaidas&hl=es-419&sa=X&redir_esc=y#v=onepage&q=tipos%20de%20paracaidas&f=false

Moreno, J. L., & Martínez, A. L. (2016). *Manual de Practicas de Tecnologia de la Fabricación.* Almería: Universidad de Almería. Recuperado el 5 de Mayo de 2016, de http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/robotica/teleoperado.htm

Robotics, A. (2005). *SoftBank Robotics*. Recuperado el 22 de Junio de 206, de https://www.ald.softbankrobotics.com/en/cool-robots/nao/find-out-more-about-nao

Robotrónica. (2016). *Aliverobots.* Recuperado el 18 de Junio de 2016, de http://aliverobots.com/nao/

Rossum, G. V. (Septiembre de 2009). *El tutorial de Python.* Recuperado el 17 de Junio de 2016, de http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf

Sanford, K. (24 de Enero de 2012). *Code Project*. Recuperado el 12 de 5 de 2016, de https://www.codeproject.com/Articles/317974/KinectDepthSmoothing

social, C. (s.f.). *Cultura Social*. Obtenido de http://www.culturasocial.org/los-10-robots-humanoides-mas-avanzados-que-existen/

Wikilibros. (Marzo de 2017). *Programación en C++*. Recuperado el 18 de Junio de 2016, de https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Introducci%C3%B3n

Zoe Falormir, L. (2006). *Robots Humanoides.* España. Obtenido de http://www.marsparachutes.com/product/productmars-mini/

ANEXO 1

PROGRAMACIÓN EN PYTHON

```
import socket
import sys
import almath
import time
import argparse
from time import sleep
from naoqi import ALProxy
PORT = 9559
IP = "192.168.0.1"
# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#ser = serial.Serial('COM4', 9600, timeout=0)
# Bind the socket to the port
server_address = ('localhost', 10008)
print >>sys.stderr, 'starting up on %s port %s' % server_address
sock.bind(server_address)
lastval11 = ''
lastval12 = ''
# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    print >>sys.stderr, 'waiting for a connection'
    connection, client_address = sock.accept()
    try:
        print >>sys.stderr, 'connection from', client_address
        motionProxy = ALProxy("ALMotion", IP, PORT)
        postureProxy = ALProxy("ALRobotPosture", IP, PORT)
        motionProxy.moveInit()

        while True:
            data = connection.recv(256)
            data = data.replace(",",".")
            data1 = data.split(';')
            #print (data11)
            val0= data1[0]# shouder right roll
```

```
val1= data1[1]# shouder left ROll
val2= data1[2]# right elbow roll
val3= data1[3]# elbow left roll
val4= data1[4]# right shoulder pitch
val5= data1[5]# left shoulder pitch
val6= data1[6]# Head roll
val7= data1[7]# hip right pitch
val8= data1[8]# hip left pitch
val9= data1[9]# hip right roll
val10= data1[10]# hip left roll
val11= data1[11]# RH state
val12= data1[12]# LH state

# CONTROL MANOS
# Izquierda
if (val12[3:7] != str(lastval12)):
            #time.sleep (0.3)
    if (val12[3:7] == 'True'):
           #print "abro"
            L = True
            if L == True:
                motionProxy.post.openHand('LHand')
                L = False
                val12[3:7] == 'True'
    if (val12[3:8] == 'False'):
           #print "cierro"
            L = False
            if L == False:
                motionProxy.post.closeHand('LHand')
                L = True
                val12[3:8] == 'False'
            time.sleep (0.01)
lastval12 = val12[3:7]
# Derecha
if (val11[3:7] != str(lastval11)):
            #time.sleep (0.3)
            if (val11[3:7] == 'True'):
                #print "abro"
                R = True
                if R == True:
```

```
                        motionProxy.post.openHand('RHand')
                        R = False
                        val11[3:7] == 'True'
            if (val11[3:8] == 'False'):
                    #print "cierro"
                    R = False
                    if R == False:
                        motionProxy.post.closeHand('RHand')
                        R = True
                        val11[3:8] == 'False'
            time.sleep (0.01)
lastval11 = val11[3:7]

#MOVIMIENTO EXTREMIDADES SUPERIORES
if val3[0:3] == 'EBL':#elbow left roll
    vald=(val3[4:10])
    vald= float(vald)
    vald=(round (vald,2))
    #print (vald)
    vald =  (-vald - 2.67)  #elbow left roll
    motionProxy.setAngles("LElbowRoll", vald, 0.2) #elbow left roll
if val2[0:3] == 'EBR':# elbow right roll
    valc=(val2[4:10])
    valc= float(valc)
    valc=(round (valc,2))
    #print (valc)
    valc =  ((valc) + 2.67) # elbow right roll
    motionProxy.setAngles("RElbowRoll", valc, 0.2) # elbow right roll
if val0[0:3] == 'SHR': #Shoulder Right
    vala=(val0[4:10])
    vala= float(vala)
    vala=(round (vala,2))
    #print (vala)
    vala =  (0.30 - abs(vala)) # shouder right roll
    motionProxy.setAngles("RShoulderRoll", vala, 0.2)#shoulder right roll
if val1[0:3] == 'SHL':#Shoulder Left
    valb=(val1[4:10])
    #print (valb)
    valb= float(valb)
    valb=(round (valb,2))
```

```
        #print (valb)
        # shouder left ROll\
        valb =  (abs(valb) - 0.30) # shouder left roll
        motionProxy.setAngles("LShoulderRoll", valb, 0.2) #shoulder left roll
if val4[0:3] == 'RSP':#Right shoulder Pitch
        vale=(val4[4:10])
        #print (valc)
        vale= float(vale)
        vale=(round (vale,2))
        #print (vale)
        vale =  (1.69 - abs(vale)) #right shoulder pitch
        motionProxy.setAngles("RShoulderPitch", vale, 0.2)
if val5[0:3] == 'LSP':#Left shoulder Pitch
        valf=(val5[4:10])
        #print (valc)
        valf= float(valf)
        valf=(round (valf,2))
        #print (valf)
        valf =  (1.69 - abs(valf))#left shoulder pitch
        motionProxy.setAngles("LShoulderPitch", valf, 0.2)

#CONTROL MUÑECA
if val6[0:3] == 'HRL': #Head Roll (mov_muñeca)
        valg=val6[4:10]
        valg= float(valg)
        valg=(round (valg,2))
        if (valg >0.20 and valg<=0.40):
           valg= 1.21+valg
           motionProxy.setAngles("LWristYaw", valg, 0.1)
           motionProxy.setAngles("RWristYaw", -valg, 0.1)
        if (valg >=0 and valg<=0.20):
           valg= 0.35
           motionProxy.setAngles("LWristYaw", valg, 0.1)
           motionProxy.setAngles("RWristYaw", -valg, 0.1)

#MOVIMIENTO EXTREMIDADES INFERIORES
motionProxy.post.setWalkArmsEnabled(False, False)
if val7[0:3] == 'RHP': #Right Hip pitch
        valh=val7[4:10]
        valh= float(valh)
```

```
            valh=(round (valh,2))
            if (valh >=-1.50 and valh<=-1.00):
               motionProxy.post.moveTo(0.1,0,0)#adelante
               time.sleep (0.01)
        if val8[0:3] == 'LHP': #Left Hip pitch
            vali=val8[4:10]
            vali= float(vali)
            vali=(round (vali,2))
            if (vali >=-1.50 and vali<=-1.00):
               motionProxy.post.moveTo(-0.1,0,0)#atras
               time.sleep (0.01)
        if val9[0:3] == 'RHR': #Right Hip roll
            valj=val9[4:10]
            valj= float(valj)
            valj=(round (valj,2))
            if (valj >=3.7 and valj <=4.0):
                motionProxy.post.moveTo(0,-0.1,0)#derecha
                time.sleep (0.01)
        if val10[0:3] == 'LHR': #Left Hip roll
            valk=val10[4:10]
            valk= float(valk)
            valk=(round (valk,2))
            if (valk >=3.7 and valk <=4.0):
               motionProxy.post.moveTo(0,0.1,0)#izquierda
               time.sleep (0.01)
finally:
    # Clean up the connection
    connection.close()
```

ANEXO 2

PROGRAMACIÓN EN C#

```
// System imports
using System;
using System.Windows.Threading;
using System.Windows.Media;
using System.Windows.Media.Media3D; // For 3D vectors
using System.IO.Ports;
using System.Threading;

// Microsoft imports
using Microsoft.Kinect;
using System.Net.Sockets;

namespace NAO_Kinect
{
    public delegate void ProcessingNewFrameEventHandler(object sender,
EventArgs e);
    public delegate void ProcessingNewSpeechEventHandler(object sender,
EventArgs e);
    public delegate void ProcessingNewTickEventHandler(object sender,
EventArgs e);

    /// <summary>
    /// This class takes a tracked body and generates useful data from it
    /// </summary>
    class Processing
    {
        /// <summary>
        /// Struct to return all relevant data to other classes
        /// </summary>
        internal struct BodyInfo
        {
            public float[] angles;
            //public string[] angulos;


            public bool RHandOpen;
            public bool LHandOpen;
            public bool noTrackedBody;

        };

        /// <summary>
        /// Variables
        /// </summary>
        private bool allowNaoUpdates = false;
        private bool invert = true;
        private string rHandStatus = "unknown";
        private string lHandStatus = "unkown";

        private string[] angulos;
        private string[] manos;
        private string[] noTracked;
        private readonly string[] invertedJointNames = { "LShoulderRoll",
"RShoulderRoll", "LElbowRoll", "RElbowRoll", "LShoulderPitch",
"RShoulderPitch" };
```

```
        private readonly string[] jointNames = { "RShoulderRoll",
"LShoulderRoll", "RElbowRoll", "LElbowRoll", "RShoulderPitch",
"LShoulderPitch" };
        private float[] offset = { 0.8f, 0.8f, -2.5f, -2.5f, -2.65f, -2.65f };
        private float[] oldAngles = new float[6];
        private static KinectInterface kinectInterface;
        private static Body trackedBody;
        BodyInfo bodyInfo;
        BodyInfo info;
        private BodyInfo UIinfo;
        private Motion naoMotion;
        ImageSource currentFrame;
        string speechStatus;
        bool speechResult;


        public event ProcessingNewFrameEventHandler pNewFrame;
        public event ProcessingNewSpeechEventHandler pNewSpeech;
        public event ProcessingNewSpeechEventHandler pNewTick;

        /// <summary>
        /// Timer
        /// </summary>
        private DispatcherTimer motionTimer = new DispatcherTimer();


             System.Net.Sockets.TcpClient clientSocket = new
System.Net.Sockets.TcpClient();

             /// <summary>
             /// Class constructor
             /// </summary>
             /// <param name="interfaceClass"> Reference to current kinect
interface </param>
             public Processing()
        {
            bodyInfo = new BodyInfo();
            bodyInfo.angles = new float[11];

            angulos = new string[11];
            manos = new string[2];
           // noTracked = new string[2];

            // Call the motion constructor
            naoMotion = new Motion();

            // Creates the kinectInterface class and registers event handlers
            kinectInterface = new KinectInterface();
            kinectInterface.start();

            kinectInterface.NewFrame += kinectInterface_NewFrame;
            kinectInterface.NewSpeech += kinectInterface_NewSpeech;

            // Create a timer for event based NAO update.
            motionTimer.Interval = new TimeSpan(0, 0, 0, 0,
(int)Math.Ceiling(1000.0 / 7));
            motionTimer.Start();

            motionTimer.Tick += motionTimer_Tick;

                    //Creacion de cliente y puerto socket
```

```
                        //msg("Client Started");
                        clientSocket.Connect("localhost", 10008);

                }

                public void connect(string ip)
            {
                naoMotion.connect(ip);

                allowNaoUpdates = true;
            }

            public void disconnect()
            {
                allowNaoUpdates = false;
            }

            public void setInvert(bool set)
            {
                invert = set;
            }

            public string getSpeechStatus()
            {
                return speechStatus;
            }

            public bool getSpeechResult()
            {
                return speechResult;
            }

            public BodyInfo getBodyInfo()
            {
                return UIinfo;
            }

            public ImageSource getFrame()
            {
                return currentFrame;
            }

            public void cleanUp()
            {
                naoMotion.removeStiffness();
            }

                /// <summary>
                /// Gets the usable angles of joints for sending to NAO
                /// </summary>
                /// <returns> struct of tupe BodyInfo </returns>
                private BodyInfo calculateAngles()
                {
                        trackedBody = kinectInterface.getBody();

                        if (trackedBody != null)
                        {
                                bodyInfo.noTrackedBody = false;
                    //noTracked[0] = string.Concat("T:", bodyInfo.noTrackedBody);

                                var shoulderCenter =
trackedBody.Joints[JointType.SpineShoulder].Position;
```

```
                            var wristLeft =
trackedBody.Joints[JointType.WristLeft].Position;
                            var wristRight =
trackedBody.Joints[JointType.WristRight].Position;

                            //var spineShoulder =
trackedBody.Joints[JointType.SpineShoulder].Position;
                            //var spineBase =
trackedBody.Joints[JointType.SpineBase].Position;

                            var elbowLeft =
trackedBody.Joints[JointType.ElbowLeft].Position;
                            var elbowRight =
trackedBody.Joints[JointType.ElbowRight].Position;

                            var shoulderLeft =
trackedBody.Joints[JointType.ShoulderLeft].Position;
                            var shoulderLeftNorm =
trackedBody.JointOrientations[JointType.ShoulderLeft].Orientation;
                            var shoulderRight =
trackedBody.Joints[JointType.ShoulderRight].Position;
                            var shoulderRightNorm =
trackedBody.JointOrientations[JointType.ShoulderRight].Orientation;

                            var hipLeft =
trackedBody.Joints[JointType.HipLeft].Position;
                            var hipRight =
trackedBody.Joints[JointType.HipRight].Position;

                var head = trackedBody.Joints[JointType.Head].Position;
                var neck = trackedBody.Joints[JointType.Neck].Position;
                var spineShoulder =
trackedBody.Joints[JointType.SpineShoulder].Position;

                var spineBase =
trackedBody.Joints[JointType.SpineBase].Position;
                var kneeLeft =
trackedBody.Joints[JointType.KneeLeft].Position;
                var kneeRight =
trackedBody.Joints[JointType.KneeRight].Position;

                var hipLeftNorm =
trackedBody.JointOrientations[JointType.HipLeft].Orientation;
                var hipRightNorm =
trackedBody.JointOrientations[JointType.HipRight].Orientation;

                var ankleLeft =
trackedBody.Joints[JointType.AnkleLeft].Position;
                var ankleRight =
trackedBody.Joints[JointType.AnkleRight].Position;

                var footLeft =
trackedBody.Joints[JointType.FootLeft].Position;
                var footRight =
trackedBody.Joints[JointType.FootRight].Position;

                            switch (trackedBody.HandRightState)
                            {
                                    case HandState.Open:
                                            bodyInfo.RHandOpen = true;
                        //bodyInfo.angles[6] = 0.5f;
```

```
                        manos[0] = string.Concat("RH:", bodyInfo.RHandOpen);

                                break;

                            case HandState.Closed:
                                bodyInfo.RHandOpen = false;
                        // bodyInfo.angles[6] = 0.1f;
                        manos[0] = string.Concat("RH:", bodyInfo.RHandOpen);
                                break;

                            case HandState.Lasso:
                                bodyInfo.RHandOpen = false;
                        //manos[2] = string.Concat("RHL:",
bodyInfo.RHandOpen);
                                break;


                        }

                        switch (trackedBody.HandLeftState)
                        {
                            case HandState.Open:
                                bodyInfo.LHandOpen = true;
                        manos[1] = string.Concat("LH:", bodyInfo.LHandOpen);
                                break;
                            case HandState.Closed:
                                bodyInfo.LHandOpen = false;
                        manos[1] = string.Concat("LH:", bodyInfo.LHandOpen);
                                break;
                            case HandState.Lasso:
                                bodyInfo.LHandOpen = false;
                        //manos[5] = string.Concat("LHL:",
bodyInfo.LHandOpen);
                                break;
                        }

                        var rollRefRight = new CameraSpacePoint();
                        rollRefRight.X = shoulderRight.X;
                        rollRefRight.Y = elbowRight.Y;
                        rollRefRight.Z = elbowRight.Z;

                        var rollRefLeft = new CameraSpacePoint();
                        rollRefLeft.X = shoulderLeft.X;
                        rollRefLeft.Y = elbowLeft.Y;
                        rollRefLeft.Z = elbowLeft.Z;

                        /*if (elbowRight.Y < shoulderRight.Y)
                {
                    // Stores the right shoulder roll in radians
                    bodyInfo.angles[0] = angleCalc3D(rollRefRight,
shoulderRight, elbowRight);
                }

                if (elbowLeft.Y < elbowRight.Y)
                {
                    // Stores the left shoulder roll in radians
                    bodyInfo.angles[1] = angleCalc3D(rollRefLeft,
shoulderLeft, elbowLeft);
                }*/
```

```
                            bodyInfo.angles[0] = getShoulderRoll(shoulderRight,
elbowRight, hipRight);

                float truncated0 = (float)(Math.Truncate((double)
bodyInfo.angles[0] * 100.0) / 100.0);
                float val0 = (float)(Math.Round((double)bodyInfo.angles[0],
2));

                angulos[0] = string.Concat("SHR:", val0);



                            bodyInfo.angles[1] = getShoulderRoll(shoulderLeft,
elbowLeft, hipLeft);

                float truncated1 =
(float)(Math.Truncate((double)bodyInfo.angles[1] * 100.0) / 100.0);
                float val1 = (float)(Math.Round((double)bodyInfo.angles[1],
2));

                angulos[1] = string.Concat("SHL:", val1);

                            // Stores the right elbow roll in radians
                            bodyInfo.angles[2] = 0 - angleCalc3D(shoulderRight,
elbowRight, wristRight);

                float truncated2 =
(float)(Math.Truncate((double)bodyInfo.angles[2] * 100.0) / 100.0);
                float val2 = (float)(Math.Round((double)bodyInfo.angles[2],
2));


                angulos[2] = string.Concat("EBR:", val2);

                            // Stores the left elbow roll in radians
                            bodyInfo.angles[3] = 0 - angleCalc3D(shoulderLeft,
elbowLeft, wristLeft);
                float truncated3 =
(float)(Math.Truncate((double)bodyInfo.angles[3] * 100.0) / 100.0);
                float val3 = (float)(Math.Round((double)bodyInfo.angles[3],
2));


                angulos[3] = string.Concat("EBL:", val3);

                            // Shoulder pitch should be same as shoulder roll
but with angleCalcYZ
                            // Stores the right shoulder pitch in radians
                            bodyInfo.angles[4] = 0 - angleCalcYZ(hipRight,
shoulderRight, elbowRight) * 1.5f;

                float truncated4 =
(float)(Math.Truncate((double)bodyInfo.angles[4] * 100.0) / 100.0);
                float val4 = (float)(Math.Round((double)bodyInfo.angles[4],
2));

                angulos[4] = string.Concat("RSP:", val4);


                            // Stores the left shoulder pitch in radians
                            bodyInfo.angles[5] = 0 - angleCalcYZ(hipLeft,
shoulderLeft, elbowLeft) * 1.5f;
```

```
                float truncated5 =
(float)(Math.Truncate((double)bodyInfo.angles[5] * 100.0) / 100.0);
                float val5 = (float)(Math.Round((double)bodyInfo.angles[5],
2));


                angulos[5] = string.Concat("LSP:", val5);

                bodyInfo.angles[6] = getHeadRoll(spineShoulder, head, neck);
                float truncated6 =
(float)(Math.Truncate((double)bodyInfo.angles[6] * 100.0) / 100.0);
                float val6 = (float)(Math.Round((double)bodyInfo.angles[6],
2));


                angulos[6] = string.Concat("HRL:", val6);

                //Hip Right Pitch
                bodyInfo.angles[7] = 0 - angleCalcYZ(kneeRight, hipRight,
ankleRight) * 1.5f;
                float truncated7 =
(float)(Math.Truncate((double)bodyInfo.angles[7] * 100.0) / 100.0);
                float val7 = (float)(Math.Round((double)bodyInfo.angles[7],
2));

                angulos[7] = string.Concat("RHP:", val7);

                //Hip Left Pitch
                bodyInfo.angles[8] = 0 - angleCalcYZ(kneeLeft, hipLeft,
ankleLeft) * 1.5f;
                float truncated8 =
(float)(Math.Truncate((double)bodyInfo.angles[8] * 100.0) / 100.0);
                float val8 = (float)(Math.Round((double)bodyInfo.angles[8],
2));

                angulos[8] = string.Concat("LHP:", val8);

                //Hip Right Roll
                bodyInfo.angles[9] = getHipRoll(kneeRight, hipRight,
footRight);
                float truncated9 =
(float)(Math.Truncate((double)bodyInfo.angles[9] * 100.0) / 100.0);
                float val9 = (float)(Math.Round((double)bodyInfo.angles[9],
2));

                angulos[9] = string.Concat("RHR:", val9);

                //Hip Left Roll
                bodyInfo.angles[10] = getHipRoll(kneeLeft, hipLeft, footLeft);
                float truncated10 =
(float)(Math.Truncate((double)bodyInfo.angles[10] * 100.0) / 100.0);
                float val10 = (float)(Math.Round((double)bodyInfo.angles[10],
2));

                angulos[10] = string.Concat("LHR:", val10);

                    }
                    else
                    {
                            bodyInfo.noTrackedBody = true;
                //noTracked[1] = string.Concat("NT:", bodyInfo.noTrackedBody);
```

```
                }

                // Envio de datos por socket

            /*
                string data="";
                foreach(float i in bodyInfo.angles){
                    data += i + ";";


                     }

            string data1 = string.Concat(data, bodyInfo.RHandOpen,";");

            //Console.WriteLine(angulos[4]);

            */

            string datax = "";

            foreach (string i in angulos)
            {
                datax += i + ";";


            }
            string datay = "";
            foreach (string i in manos)
            {
                datay += i + ";";


            }

            string data1 = string.Concat(datax,datay,";");

            Console.WriteLine(data1);

                    NetworkStream serverStream = clientSocket.GetStream();
            byte[] outStream = System.Text.Encoding.ASCII.GetBytes(data1);

              serverStream.Write(outStream, 0, outStream.Length);
            serverStream.Flush();

                    return bodyInfo;


              }


              // Calculates angle between a<-b and b->c for situation a->b->c
              private static float getShoulderRoll(CameraSpacePoint shoulder,
CameraSpacePoint elbow, CameraSpacePoint hip)
        {
            CameraSpacePoint xzRef = new CameraSpacePoint();
            xzRef.X = shoulder.X;
            xzRef.Y = elbow.Y;
            xzRef.Z = elbow.Z;

            var anglexz = angleCalcXZ(xzRef, shoulder, elbow);
            var anglexy = angleCalcXY(hip, shoulder, elbow);
```

```
            return (float)(anglexz * anglexy)/1.2f;
        }

        private static float getHeadRoll(CameraSpacePoint spineShoulder,
CameraSpacePoint head, CameraSpacePoint neck)
        {
            CameraSpacePoint xzRef = new CameraSpacePoint();
            xzRef.X = spineShoulder.X;
            xzRef.Y = head.Y;
            xzRef.Z = head.Z;

            var anglexz = angleCalcXZ(xzRef, spineShoulder, head);
            var anglexy = angleCalcXY(neck, spineShoulder, head);

            return (float)(anglexz * anglexy) / 1.2f;
        }

        private static float getHipRoll(CameraSpacePoint knee,
CameraSpacePoint hip, CameraSpacePoint foot)
        {
            CameraSpacePoint xzRef = new CameraSpacePoint();
            xzRef.X = knee.X;
            xzRef.Y = hip.Y;
            xzRef.Z = hip.Z;

            var anglexz = angleCalcXZ(xzRef, knee, hip);
            var anglexy = angleCalcXY(foot, knee, hip);

            return (float)(anglexz * anglexy) / 1.2f;
        }


        // Calculates angle between a<-b and b->c for situation a->b->c
        private static float angleCalc3D(CameraSpacePoint a, CameraSpacePoint
b, CameraSpacePoint c)
        {
            var ba = new Vector3D(a.X - b.X, a.Y - b.Y, a.Z - b.Z);
            var bc = new Vector3D(c.X - b.X, c.Y - b.Y, c.Z - b.Z);

            var angle = (float)Vector3D.AngleBetween(ba, bc); // degrees
            return (float)(Math.PI / 180) * angle; // radians
        }



        // Calculates angle between a<-b and b->c for situation a->b->c
        private static float angleCalcXZ(CameraSpacePoint a, CameraSpacePoint
b, CameraSpacePoint c)
        {
            var ba = new Vector3D(a.X - b.X, 0, a.Z - b.Z);
            var bc = new Vector3D(c.X - b.X, 0, c.Z - b.Z);

            var angle = (float)Vector3D.AngleBetween(ba, bc); // degrees
            return (float)(Math.PI / 180) * angle; // radians
        }

        // Calculates angle between a<-b and b->c for situation a->b->c on XY
plane only
        private static float angleCalcXY(CameraSpacePoint a, CameraSpacePoint
b, CameraSpacePoint c)
        {
            var ba = new Vector3D(a.X - b.X, a.Y - b.Y, 0);
```

```
            var bc = new Vector3D(c.X - b.X, c.Y - b.Y, 0);

            var angle = (float)Vector3D.AngleBetween(ba, bc); // degrees

            return (float)(Math.PI / 180) * angle; // radians
        }

        // Calculates angle between a<-b and b->c for situation a->b->c on YZ
plane only
        private static float angleCalcYZ(CameraSpacePoint a, CameraSpacePoint
b, CameraSpacePoint c)
        {
            var ba = new Vector3D(0, a.Y - b.Y, a.Z - b.Z);
            var bc = new Vector3D(0, c.Y - b.Y, c.Z - b.Z);

            var angle = (float)Vector3D.AngleBetween(ba, bc); // degrees
            return (float)(Math.PI / 180) * angle; // radians
        }

        /// ******************************************************
        ///
        ///                     TIMER
        ///
        /// ******************************************************

        /// <summary>
        /// Timer to rate limit NAO joint updates
        /// </summary>
        /// <param name="sender"> Object that generated the event </param>
        /// <param name="e"> Any additional arguments </param>
        private void motionTimer_Tick(object sender, EventArgs e)
        {
            // Gets array of info from bodyProcessing
            info = calculateAngles();

            if (!info.noTrackedBody)
            {

                // Generate calibrated angles
                for (var x = 0; x < 6; x++)
                {
                    info.angles[x] = info.angles[x] - offset[x]; // adjustment
to work with NAO robot angles
                }

                if (info.angles[4] < -2.0f) info.angles[4] = -2.0f;
                if (info.angles[5] < -2.0f) info.angles[5] = -2.0f;

                if (info.angles[4] > 2.0f) info.angles[4] = 2.0f;
                if (info.angles[5] > 2.0f) info.angles[5] = 2.0f;



                // Check if updates should be sent to NAO
                if (allowNaoUpdates)
                {
                    // Check to make sure that angle has changed enough to
send new angle and update angle if it has
                    for (var x = 0; x < 6; x++)
                    {
                        if ((Math.Abs(oldAngles[x] - info.angles[x]) > .1 ||
Math.Abs(info.angles[x] - info.angles[x]) < .1))
```

```
                        {
                            oldAngles[x] = info.angles[x];
                            updateNAO(info.angles[x], invert ?
invertedJointNames[x] : jointNames[x]);
                        }
                    }

                    // update right hand
                    switch (info.RHandOpen)
                    {
                        case true:
                            if (rHandStatus == "open")
                            {
                                break;
                            }
                            rHandStatus = "open";
                            if (invert)
                            {
                                naoMotion.openHand("LHand");
                                break;
                            }
                            naoMotion.openHand("RHand");
                            break;
                        case false:
                            if (rHandStatus == "closed")
                            {
                                break;
                            }
                            rHandStatus = "closed";
                            if (invert)
                            {
                                naoMotion.closeHand("LHand");

                                break;
                            }
                            naoMotion.closeHand("RHand");
                            break;
                    }

                    // update left hand
                    switch (info.LHandOpen)
                    {
                        case true:
                            if (lHandStatus == "open")
                            {
                                break;
                            }
                            lHandStatus = "open";
                            if (invert)
                            {
                                naoMotion.openHand("RHand");
                                break;
                            }
                            naoMotion.openHand("LHand");
                            break;
                        case false:
                            if (lHandStatus == "closed")
                            {
                                break;
                            }
                            lHandStatus = "closed";
                            if (invert)
```

```
                            {
                                naoMotion.closeHand("RHand");

                                break;
                            }
                            naoMotion.closeHand("LHand");
                            break;
                        }

                    }
                }

                if (pNewTick != null)
                {
                    UIinfo = info;
                    pNewTick(this, EventArgs.Empty);
                }
            }

            /// ******************************************************
            ///
            ///                     KINECT EVENTS
            ///
            /// ******************************************************

            /// <summary>
            /// Event handler for new frames created by the kinectBody class
            /// </summary>
            /// <param name="sender"> Object that generated the event </param>
            /// <param name="e"> Any additional arguments </param>
            private void kinectInterface_NewFrame(object sender, EventArgs e)
            {
                // Gets the image from kinectInterface class and updates the image
in the UI
                currentFrame = kinectInterface.getImage();

                if (pNewFrame != null)
                {
                    pNewFrame(this, EventArgs.Empty);
                }
            }

            /// <summary>
            /// Event handler for new frames created by the kinectBody class
            /// </summary>
            /// <param name="sender"> Object that generated the event </param>
            /// <param name="e"> Any additional arguments </param>
            private void kinectInterface_NewSpeech(object sender, EventArgs e)
            {
                var result = kinectInterface.getResult();
                var semanticResult = kinectInterface.getSemanticResult();
                var confidence = kinectInterface.getConfidence();

                // If confidence of recognized speech is greater than 60%
                if (confidence > 0.6)
                {
                    // Debug output, tells what phrase was recongnized and the
confidence
                    speechStatus = "Recognized: " + result + " \nConfidence: " +
confidence;

                    if(semanticResult == "on")
```

```
                {
                    speechResult = true;
                }

                if(semanticResult == "off")
                {
                    speechResult = false;
                }
            }
            else // Else say that it was rejected and confidence
            {
                speechStatus = "Rejected " + " \nConfidence: " + confidence;
            }

            if (pNewSpeech != null)
            {
                pNewSpeech(this, EventArgs.Empty);
            }
        }


        /// ********************************************************
        ///
        ///                     NAO METHODS
        ///
        /// ********************************************************

        private void updateNAO(float angle, string joint)
        {
            // RShoulderRoll and RElbowRoll require inverted angles
            if (joint == "RShoulderRoll" || joint == "LElbowRoll")
            {
                // Invert the angle
                angle = (0 - angle);
            }

            // Check for error when moving joint
            if (!naoMotion.moveJoint(angle, joint))
            {
                //debug3.Text = "Exception occured when communicating with NAO
check C:\\NAO Motion\\ for details";
            }

            if (joint == "RElbowRoll")
            {
               // Console.WriteLine(joint + ": " + angle);


                    }
              }
    }
}
```