CAPTURA DE MOVIMIENTO UTILIZANDO EL KINECT PARA EL CONTROL DE UNA PLATAFORMA ROBOTICA CONTROLADA DE FORMA REMOTA POR MEDIO DE SEGUIMIENTO DE LOS PUNTOS DE ARTICULACIÓN DEL CUERPO

OLGA PATRICIA OSORIO
FRANCISCO LEANDRO PEÑA

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA, FÍSICA Y DE CIENCIAS DE LA COMPUTACIÓN
PROGRAMA DE INGENIERÍA ELECTRÓNICA
PEREIRA, COLOMBIA
2015

CAPTURA DE MOVIMIENTO UTILIZANDO EL KINECT PARA EL CONTROL DE UNA PLATAFORMA ROBOTICA CONTROLADA DE FORMA REMOTA POR MEDIO DE SEGUIMIENTO DE LOS PUNTOS DE ARTICULACIÓN DEL CUERPO

OLGA PATRICIA OSORIO

FRANCISCO LEANDRO PEÑA

Trabajo presentado como requisito para optar al título de

Ingeniero Electrónico

Director: ING. ARLEY BEJARANO MARTINEZ

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA, FÍSICA Y DE CIENCIAS DE LA COMPUTACIÓN

PROGRAMA DE INGENIERÍA ELECTRÓNICA

PEREIRA, COLOMBIA

2015

Nota de aceptación

_____

Firma del presidente del jurado

_____

Firma del jurado

_____

Firma del jurado

Pereira, Febrero de 2015.

AGRADECIMIENTOS

A nuestro Padre Celestial por derramar sus bendiciones sobre nuestras vidas!

A nuestras familias por su apoyo incondicional y su amor en todas las etapas de nuestra formación y vida.

A nuestro compañero William Alexis Ramírez por su amistad en todo momento durante nuestra formación académica.

Al Ingeniero Arley Bejarano por su ayuda e interés en este proyecto, por sus aportes valiosos en la construcción de este trabajo.

A todos los profesores que hicieron parte de nuestra formación por los conocimientos dados en lo profesional y personal.

GRACIAS!

RESUMEN

La apertura de mercados y la visualización de una calidad de vida más amena y accesible a todos, hace que cada día crezcan las líneas de tecnología con miras a convertirse en aliadas del ser humano para toda actividad y más si esta coloca en riesgo la preservación de la especie humana como tal.

Situaciones donde se liberen gases, espacios confinados, terrenos minados y desastres naturales; es ahí donde la interacción natural entre el hombre y la máquina se hacen necesaria; se convierte el robot en una aplicación moderna para interactuar en diversas circunstancias.

A lo largo del tiempo se han desarrollado métodos de captura de movimiento los cuales han sido empleados en áreas como la medicina, la educación, la seguridad entre otros. De esta manera se presenta en el mercado el sensor Kinect, dispositivo diseñado con tal propósito inicialmente, lo cual se fue diversificando a lo largo que fueron presentándose códigos abiertos para acceder y focalizar más aplicaciones con él.

Este proyecto realizó la integración del Kinect con una línea bastante robusta como es la robótica, utilizando el software ROS, haciendo uso de controladores y paquetes para obtener la esqueletización y visualización de joints.

Se implementó el paquete Skeleton_Marker que tiene como característica reconocer un usuario y realizar la esqueletización generando un solo ID, también se hizo uso del controlador Nite para obtener la imagen de profundidad que garantizó la estabilidad de los joints.

Una vez se accedió a la imagen, se pasó a la etapa de creación de un nuevo paquete para recibir la información generada por los puntos sensados; para esta tarea se implementó Transform Frame de ROS (TF), que hace relación a una transformación de los datos generados para ser enviados; en esta secuencia se utilizó los mensajes de ROS (listener) para enviar y recibir la información de un paquete a otro.

Para lograr una visión del comportamiento del joint se implementó por código la toma de 100 muestras de cada punto, vistas por terminal en los tres ejes coordenados; y 10 muestras reales las cuales se hicieron con un medidor laser, con el fin de relacionar las medidas generadas por el Kinect y un instrumento de medida.

Una vez se obtuvo todos los datos, se realizó un tratamiento estadístico para presentar el grado de error a nivel de joints y global tomando el esqueleto.

TABLA DE CONTENIDO

1. INTRODUCCION 1

1.1 DEFINICIÓN DEL PROBLEMA 2

1.2 JUSTIFICACIÓN 4

1.3 OBJETIVOS 5

1.3.1 OBJETIVO GENERAL 5

1.3.2 OBJETIVOS ESPECIFICOS 5

1.4 MARCO DE ANTECEDENTES 6

2. MARCO TEORICO 16

2.1 MICROSOFT KINECT 16

2.1.1 Arquitectura del Kinect 17

2.1.2 Cámara de profundidad. 18

2.1.3 Reconstrucción de objetos. 19

2.1.4 Triangulación. 19

2.1.5 Imágenes de profundidad. 21

2.1.6 Detección y seguimiento de usuario 22

2.1.7 Rastreo del esqueleto 23

2.2 BIBLIOTECAS LIBRES DE KINECT 26

3. CAPTURA DE MOVIMIENTO 30

3.1 IMAGEN DIGITAL 30

3.2 IMÁGENES A COLOR 30

3.2.1 Modelos de color 31

3.3 CAPTURA DE MOVIMIENTO 31

3.3.1 El modelo de barras de Chen y Lee. 33

3.3.2 Captura de movimientos electromecánica 33

3.3.3 Captura de movimiento electromagnética. 34

3.3.4 Captura óptica de movimiento 34

3.3.5 Mediante indicadores activos. 35

3.3.6 Mediante indicadores activos modulados en el tiempo 36

3.3.7 Captura mediante fibra óptica 36

3.3.8 Captura mediante ultrasonidos. 37
3.3.9 Captura mediante sistemas inerciales. 37
3.3.10 Captura óptica de movimiento pos substracción de video. 37
4. SOFTWARE ROBOT OPERATING SYSTEM ROS 39
4.1 CONCEPTOS BASICOS 40
4.1.1 Sistemas de archivos 40
4.1.2 Componentes de los paquetes 42
4.1.3 Compilación 42
4.1.4 Nodos. 43
4.1.5 Topics. 45
4.1.6. Servicios. 46
4.1.7 Mensajes. 47
4.1.8 Master 48
4.1.9 Lanzadores 49
4.2 HERRAMIENTAS DE LÍNEAS DE COMANDOS. 50
4.3 TRABAJANDO EN ROS. 52
4.3.1 Creación de un Workspace. 52
4.3.2. Creación de un directorio para un nuevo paquete 54
4.3.3 Creación de un nuevo paquete en ROS 56
4.3.4 Inicializando ROS 60
4.3.5 Creación de MSG Y SRV (mensajes y servicios) 61
4.3.6 Manejo del Kinect en ROS. 66
4.3.7. Configuración de Rviz en ROS 78
4.4 PAQUETE OPENNI DE ROS. 79
4.4.1 Librerías. 79
4.4.2 Paquete tf (transform). 86
4.5. TRANSFORM FRAME (TF) 87
Diferentes tipos de puntos. 89
4.6 TRANSFORMACIONES. 89
4.7. ROTACIÓN EN ROS. 91
Rotación alrededor de un eje fijo 91
4.8 COMPARACIÓN ENTRE LOS MÉTODOS DE ROTACIÓN. 99
5. MEDIDAS 101

5.1 POSICIÓN INCORRECTA DE ARTICULACIONES 101
5.2 PAQUETE TF BROADCASTER Y LISTENER..... 111
5.2.1 TF Broadcaster 111
5.2.2 TF Listener..... 114
5.3 PROMEDIO DE DATOS..... 117
5.4 TOMA DE MEDIDAS..... 117
6. RESULTADOS..... 125
CONCLUSIONES..... 135
TRABAJOS FUTUROS..... 136
BIBLIOGRAFIA..... 137

LISTADO DE FIGURAS

Figura 1. Sistema Catrasys [6]. 6
Figura 2. Estructura general del sistema [8]. 7
Figura 3. Arquitectura del sistema [11]. 8
Figura 4. Manipulación de equipos [14]. 9
Figura 5. Historial de movimiento de un usuario usando tecnología Kinect [17]. 10
Figura 6. Usuario en terapia de rehabilitación, haciendo uso de video captura [21]. 11
Figura 7. Gestos ejecutados por el usuario [24]. 12
Figura 8. Perspectiva del espacio físico para la captura de movimiento [27]. 13
Figura 9. Captura de movimiento [29]. 14
Figura 10. Archivo de captura de movimiento y sus graficas [32]. 15
Figura 11.Componentes del Kinect [33]. 16
Figura 12. Diagrama del hardware de Kinect [35]. 17
Figura 13. Cámaras de Kinect [36]. 18
Figura 14. Triangulación en visión estero [38]. 19
Figura 15. Imagen que muestra la proyección de puntos sobre una superficie [39]. 20
Figura 16. Proyección de puntos s [40]. 20
Figura 17. Imagen de profundidad capturada por la cámara [41]. 21
Figura 18. Δx para el punto proyectado [42]. 21
Figura 19. Proceso de segmentación y obtención de articulaciones a partir de imágenes de profundidad [43]. 22
Figura 20. Articulaciones del cuerpo. [44]. 23
Figura 21. Esqueleto virtual de OpenNI/NITE [45]. 24
Figura 22. Configuración de la escena del esqueleto en OpenNI/NITE. Las articulaciones se encuentran en milímetros con respecto al origen del sistema de coordenadas (posición del Kinect) y en espejo (el eje X está invertido para el lado izquierdo del esqueleto corresponda al lado derecho del cuerpo y viceversa). Kinect mira hacia el lado positivo del eje Z. [46]. 25
Figura 23. Interacción de software y hardware con la aplicación [49]. 27
Figura 24. Estructura del framework OpenNI [52]. 29
Figura 25. Análisis Biomecánico [54]. 32
Figura 26. Modelo geométrico del hemisferio superior del cuerpo humano [57]. 33
Figura 27. Traje utilizado en captura de movimiento electromecánico [58]. 34
Figura 28. Captura de movimiento de la cara usando indicadores pasivos [59]. 35
Figura 29. Traje especial con indicadores activos [60]. 35
Figura 30. Guante de fibra óptica [61]. 36
Figura 31. Traje con sensores inerciales y unidad motora [62]. 37
Figura 32. Captura de movimiento mediante substracción de video [65]. 38
Figura 33. Relación ROS y SO [66]. 39
Figura 34. Paquetes [69]. 40

Figura 35. Pilas [70]. 41
Figura 36. Repositorios [72]. 42
Figura 37. Nodos [75]. 43
Figura 38. Topics [77]. 46
Figura 39. Servicios [79]. 46
Figura 40. Nodos [81]. 48
Figura 41. Funcionamiento del Master [82]. 49
Figura 42. Creación de un workspace. 52
Figura 43. Verificación de workspace. 53
Figura 44. Verificación de Setup. 54
Figura 45. Creación de una dirección. 55
Figura 46. Verificación de la creación de dirección de un paquete en ROS. 55
Figura 47. Creación de un paquete en ROS. 56
Figura 48. Creación de archivos que componen un paquete en ROS. 57
Figura 49. Creación de paquete. 57
Figura 50. Archivos que componen el paquete en ROS. 58
Figura 51. Verificación de perfiles. 59
Figura 52. Verificación de dependencias. 59
Figura 53. Roscore. 60
Figura 54. Código para listar nodos en terminal. 60
Figura 55. Vista en pantalla de nodos activos. 61
Figura 56. Creación de un mensaje en ROS. 62
Figura 57. Creación de un servicio en ROS. 62
Figura 58. Carpetas msg y srv. 63
Figura 59. Mensaje cargado en la carpeta msg. 64
Figura 60. Gedit del mensaje cargado en msg. 64
Figura 61. Servicio cargado en la carpeta srv. 65
Figura 62. Gedit del servicio cargado. 65
Figura 63. Mensaje en pantalla. 66
Figura 64. Lanzamiento de OpenNI_Launch. 67
Figura 65. Imagen de paridad. 67
Figura 66. Lanzamiento para imagen RGB. 68
Figura 67. Imágenes obtenidas con Openni camera de Kinect. 68
Figura 68. Imágenes obtenidas con Openni de Kinect. 69
Figura 69. Imágenes obtenidas con Openni de Kinect. 69
Figura 70. Código para activar el tracker de Kinect. 70
Figura 71. Carpeta creada para guardar el controlador NITE. 70
Figura 72. Controlador NITE. 71
Figura 73. Instalación del Controlador NITE. 71
Figura 74. Sample Player NITE. 72
Figura 75. Punto de referencia del jugador. 73
Figura 76. Seguimiento a puntos de articulación. 73

Figura 77. Identificación de un jugador. 74
Figura 78. Identificación de varias siluetas en Kinect. 75
Figura 79. Instalación del paquete Skeleton_Marker. 75
Figura 80. Configuración del Skeleton_Marker. 76
Figura 81. Lanzamiento del paquete Skeleton_Marker. 76
Figura 82. Visualización del Nite y Skeleton_Marker. 76
Figura 83. Puntos de esqueletización utilizando Skeleton_Marker. 77
Figura 84. Visualización del Skeleton_Marker. 77
Figura 85. Rviz en ROS. 78
Figura 86. Librería Package de ROS [84]. 79
Figura 87. Archivos dependientes de la librería Package [85]. 79
Figura 88. Dependencias de la librería XnCppWrapper.h [86]. 80
Figura 89. Dependencias directas de la librería XnCppWrapper.h [87]. 80
Figura 90. Estructura Jerárquica. 82
Figura 91. Características de la imagen de profundidad. Las cruces amarillas indican que el pixel ha sido clasificado, y los rojos indican el desplazamiento [89]. 84
Figura 92. Randomized Decision Forest, Constituye un conjunto de árboles, cada árbol se compone de dos nodos azul que son de división y los verdes que son hoja; Las flechas rojas indican el camino que pueden tomar una entrada en particular [90]. 84
Figura 93. Paquete tf de ROS en un robot [92]. 86
Figura 94. Árbol general tf transform [93]. 88
Figura 95. Relación entre frame y puntos [94]. 89
Figura 96. Rotación en R^2 , donde un cuadrado gira alrededor del origen a través de θ [95]. 91
Figura 97. Ángulos de Euler [96]. 93
Figura 98. Matriz de transformación [97] 93
Figura 99. Los vectores (q1,q2,q3) y (x,y,z) están en el plano de la figura y el plano Q es norma a éste [98]. 94
Figura 100. Representación geométrica de los cuaterniones q, r y r' [99] 94
Figura 101. Los cuaterniones puros q, p y n tienen normal igual a 1 y la dirección de n, siendo ortogonal a las de q y p, está en la línea de intersección de los planos Q y P. Las direcciones de q y p se han escogido de tal manera que Φ es menor que 90°. [100]. 97
Figura 102. TF y Cuaternion en paquete OpenNI. 100
Figura 103. Posición incorrecta de los pies. 101
Figura 104. Posición incorrecta de la mano derecha. 102
Figura 105. Posición incorrecta de las manos. 102
Figura 106. Error de posicionamiento frente a la cámara. 103
Figura 107. Presentación de errores múltiples (Manos y pies). 103
Figura 108. Presentación de errores múltiples (Manos y pies). 104
Figura 109. Presentación de errores múltiples (Manos y pies). 104
Figura 110. Corrección de la posición. 105
Figura 111. Posicionamiento erróneo para OpenNI_Tracker. 105

Figura 112. Posición de brazo oculto. 106
Figura 113. Posición de pie oculto 106
Figura 114. Prueba de torso y hombros. 107
Figura 115. Referenciación de las piernas. 107
Figura 116. Brazos estables. 108
Figura 117. Brazos inestables. 108
Figura 118. Pies estables. 109
Figura 119. Calibración del punto cero. 109
Figura 120. Posición manos inestables. 110
Figura 121. Posición manos estables. 110
Figura 122. Posición manos inestable. 111
Figura 123. Pantallazo de coordenadas. 117
Figura 124. Medidor de distancia laser [102]. 118
Figura 125. Dispositivo de nivel laser [103]. 119
Figura 126. Punto cero entre dispositivos. 120
Figura 127. Visual de punto cero. 121
Figura 128. Rayo lateral en relación con Kinect. 122
Figura 129. Identificación de rayo emitido y ejes X y Y. 122
Figura 130. Proyector laser y proyección de ejes X y Y. 123
Figura 131. Identificación en usuario de ejes coordenados. 124
Figura 132. Visual del esqueleto, datos y promedio. 125
Figura 133. Maniquí usado para toma de datos. 126
Figura 134. Distribución Gaussiana para el hombro. 130
Figura 135. Parámetros de campana de Gauss [104]. 130
Figura 136. Valores para campana de Gauss del codo derecho. 131
Figura 137. Valor de la media inferior a cero para el joint cadera izquierda. 132
Figura 138. Valor de la media superior a cero para el joint cabeza. 132
Figura 139. Comportamiento de los datos con desviación estándar pequeña. 133
Figura 140.Comportamiento de la desviación estándar de valor alto para el joint cuello. 133

LISTADO DE TABLAS

Tabla 1. Equivalencia de tipos de datos. 48
Tabla 2. Comparación entre los métodos de rotación. 99
Tabla 3. Vector promedio medida con láser. 127
Tabla 4. Vector promedio medida Kinect. 128
Tabla 5. Error componentes vector. 129

LISTADO DE ECUACIONES

Ecuación 1. Característica f para cada pixel x de la imagen de entrada 26
Ecuación 2. Shotton..... 84
Ecuación 3. Randomized Decision Forest..... 85
Ecuación 4. Desplazamiento de un objeto dentro de un frame. 90
Ecuación 5. Desplazamiento angular. 91
Ecuación 6. Velocidad angular..... 91
Ecuación 7. Aceleración Tangencial. 92
Ecuación 8. Movimiento de Euler. 93
Ecuación 9. Ángulos de Euler. 93
Ecuación 10. Rotación en cuaterniones. 95
Ecuación 11. Producto escalar y longitud del vector. 95
Ecuación 12. Norma de q..... 95
Ecuación 13. r' y r. 96
Ecuación 14. Error RMS. 126
Ecuación 15. Desviación estándar. 129
Ecuación 16. Media. 130

LISTADO DE ANEXOS

Anexo 1. Código para lectura de Joints (100 muestras) y su promedio..... 148
Anexo 2. Gaussianas para datos reales..... 151
Anexo 3. Maniquí utilizado..... 154

1. INTRODUCCION

Durante décadas la captura de movimiento ha sido objeto de estudio, pero después de los años 70's ha tenido innumerables desarrollos debido al avance de nuevas tecnologías y su integración con las mismas la hace hoy día un campo de exploración bastante amplio en la educación, la salud, la seguridad entre otros.

Cabe destacar que la captura de movimiento consiste en poder tomar, registrar y guardar los datos generados debido al movimiento de un usuario para su análisis posterior. Esto se ha aplicado a articulaciones del cuerpo humano que emiten movimientos estables y de fácil captura como son: la cabeza, el torso, la cadera, los brazos y piernas; haciendo exclusiones como la cara y los dedos por poseer características de movimiento muy sutiles y pequeños lo que ha hecho más difícil su captura.

Debido al desarrollo de nuevas tecnologías y la integración de capturar movimiento a las cámaras, los teléfonos celulares, sistemas infrarrojos y sensores, se ha globalizado la tendencia y ha cobrado la importancia como aplicativo; Es así como el sensor Kinect diseñado para ser utilizado en video juegos, cobra importancia al liberar controladores cuyas versiones son estables permitiendo así la conexión USB del dispositivo a un computador.

Se convierte así el Kinect en un atractivo para desarrolladores por poseer cámara RGB, una cámara de profundidad y generar imágenes en 3D de la escena, para realizar tratamientos y aplicaciones de la misma.

En el presente trabajo se hace uso del Kinect utilizando el software de robótica llamado Robot Operating System (ROS) que permite hacer uso del dispositivo; utilizando el paquete Skeleton_Marker para la esqueletización, el controlador Nite para la imagen de profundidad t el paquete de Transform Frame (TF) para los datos.

El documento se encuentra dividido en seis capítulos, el primero de ellos hace relación a la definición del problema y su fundamentación; el segundo capítulo reúne toda la información del dispositivo Kinect, su funcionamiento, controladores y bibliotecas, el tercero define la captura de movimiento y los métodos desarrollados a través del tiempo para capturar movimiento; el cuarto trató de Robot Operating System (ROS) su composición y los paquetes utilizados del mismo; el quinto es medidas donde se mostró las posiciones correctas e incorrectas frente al dispositivo, los datos adquiridos y los instrumentos de medida y el sexto y último es la presentación de resultados.

1.1 DEFINICIÓN DEL PROBLEMA

En la actualidad debido al crecimiento urbano y poblacional surgen necesidades de entorno y desarrollo; es así como la línea de la robótica intenta fusionar lo electromecánico y el software para proporcionar soluciones a problemas de interacción hombre máquina buscando mejorar la operatividad y precisión de movimientos en robots y equipos en aplicaciones donde el operador debe controlar a distancia como son: áreas de difícil acceso, espacios contaminados, confinados, poblaciones donde se presenten desastres naturales o simplemente donde el operador no tiene la capacidad de ejecutar labores de forma autónoma y requiere apoyo con equipos como personas con discapacidades.

A través de los años se han implementado métodos para la captura de movimiento entre ellos se tienen:

Captura óptica de movimiento por sustracción de video.

Esta técnica consiste en tomar una imagen de la escena sin movimiento, la cual se le asigna el nombre de fondo, a partir de ahí, restársela a cada nueva imagen capturada en la escena. Esta resta se realizando operando píxel por píxel, de una de estas imágenes, con sus respectivos píxeles de la otra imagen. Cuando los píxeles de una y otra imagen coinciden, la substracción entre valores idénticos resulta cero, lo que traducido a color es el negro; y en caso contrario aparecen otros tonos.

Las fuentes más comunes de error en este tipo de captura, son los cambios de iluminación y los movimientos de la cámara, es decir en un espacio donde ingrese luz natural, es probable que los cambios de iluminación de las diferentes horas del día generen problemas. Otra situación que puede afectar es cuando la cámara sufre algún tipo de movimiento o vibración lo que hace que la técnica no sea altamente confiable [1].

Sistema de captura de movimiento magnética.

Estos sistemas emplean sensores y sistemas de control electrónico, para rastrear el movimiento, estos sensores poseen cables que van conectados directamente a un computador. Tiene consecuencias al momento de realizar el desplazamiento el usuario ya que está restringido por los cables [2].

Sistema de captura de movimiento mecánica.

Esta técnica emplea hardware adicional como guantes y diferentes estructuras que son colocadas sobre el cuerpo del actor. Los movimientos son rastreados por el hardware. Tiene problemas con el peso de los trajes usados creando restricciones que infringe el desplazamiento [3].

Sistemas de captura de movimiento ópticos.

Existen dos tecnologías principales en los sistemas de captura de movimiento ópticos: sistemas reflectivos y sistemas basados en LED´s. Su principal falencia es el corto alcance al momento de realizar la toma de datos [4].

En este contexto es necesario realizar la captura de movimiento con un dispositivo económico, que tenga robustez frente a cambios en el ambiente (luz, campos magnéticos, entre otros). Además de realizar captura en 3D de los movimientos de un humano para ser procesados y transmitidos. Para una futura integración al macro proyecto del grupo de Investigación de Ingeniería Electrónica (GIIE), como es una plataforma robótica controlada de forma remota por medio de la captura del movimiento.

1.2 JUSTIFICACIÓN

Las plantas, los animales y los seres humanos poseen patrones de movimiento tanto corporales como internos, voluntarios o involuntarios; es así como analizar el movimiento de una persona toma la forma de interés o agrado de muchos ya que intervienen no solo partes del cuerpo de fácil identificación como de aquellas que requieren una interpretación más compleja como son los movimientos de la cara o de sus dedos.

Los seres humanos poseen un conjunto de variables que se pueden determinar entre ellos tenemos el movimiento, el cual difiere en ejecución, longitud y forma dependiendo de quién lo esté ejecutando. Las plataformas robóticas desde un principio han utilizado controles remotos trabajando con la trama de datos que emite y generando una simplicidad en la relación hombre máquina.

Una de las características en las personas que sobrepasa todo control remoto es su fluidez al momento de moverse logrando ser una pieza de control y adaptabilidad de cualquier dispositivo. Sobrepasando el uso de cualquier control remoto y sustituyéndolo por movimientos naturales de cada ser humano para lograr adaptarlo a múltiples ambientes. Por tal motivo en una plataforma robótica se incrementa su facilidad en gran número debido a que el centro de estudio es el movimiento generado espontáneamente por una persona y no utilizando mandos.

Es así como el Kinect se convierte en parte fundamental para una plataforma robótica controlada de forma remota, ya que es un dispositivo que captura imagen, profundidad y detección de personas en tiempo real, brindando información de espacio. Esta información se convierte en interés de estudio porque con ella se puede determinar movimientos de una persona a través de sus cámaras y esta tecnología se puede implementar en un robot para facilitar su operación convirtiendo los movimientos del cuerpo en comandos de movimiento para el robot o se puede utilizar para incursionar en ambientes no aptos para un ser humano y los cuales puedan poner en peligro su vida, así como entornos que tengan un grado de incertidumbre en cuanto a condiciones ambientales e identificar personas incluso en la oscuridad. Hoy día se evidencian catástrofes de tipo ambiental y natural como son: los terremotos, los tsunamis, tornados entre otros que colocan en peligro la vida humana, cuyos ambientes hostiles pueden dañar o quitarle la vida a alguien y dispositivos de este tipo pueden ayudar a preservar la calidad de vida.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Visualizar en una interfaz gráfica las coordenadas X, Y y Z de los puntos sensados de articulaciones del cuerpo detectadas por Kinect.

1.3.2 OBJETIVOS ESPECIFICOS

Crear un paquete en ROS con los nodos necesarios que permita capturar, y visualizar los puntos de articulación del cuerpo humano.

Consultar y seleccionar el controlador para Kinect que proporcione más versatilidad en la captura de información.

Diseñar una interfaz gráfica para la presentación de los puntos de articulación propios del Kinect.

Organizar la información de coordenadas de articulaciones del cuerpo obtenidas por Kinect para que pueda ser trasmitida, requerida desde otro paquete o nodo en ROS.

1.4 MARCO DE ANTECEDENTES

Se encontró en diversas universidades de España implementaciones utilizando el dispositivo Kinect:

La Universidad Carlos III de Madrid realizó un estudio biomecánico analizando el desplazamiento de las personas, en función del calzado y la velocidad del individuo. Realizando la captura del movimiento por dos técnicas de vanguardia en la adquisición de datos como son: el primero de ellos es CATRASYS ("Cassino tracking system") el cual está enfocado en la robótica y el segundo, la tecnología del Kinect para seguir la forma humana.

En ambos sistemas se efectúan medidas biomecánicas durante la tarea de caminar, para cuatro personas distintas y diferentes condiciones de marcha, en cuanto a velocidad y tipo de calzado, como se puede observar en la figura 1[5].

Figura 1. Sistema Catrasys [6].

Se implementó en la Universidad de Valéncia un sistema de interacción basado en realidad aumentada que permite conservar a un personaje virtual controlado por un actor en tiempo real con el público asistente a través de una pantalla de gran formato a modo de espejo aumentado. El sistema integra imágenes de videos reales y otros objetos animados incorporados como apoyo en la conversación, para su implementación se utilizó la combinación de dos tecnologías: Kinect para captura de movimientos corporales, mapa de profundidad e imágenes, un giróscopo para detección de movimiento de la cabeza y algoritmos de control para gestionar las expresiones. Así como lo muestra la figura 2 [7].

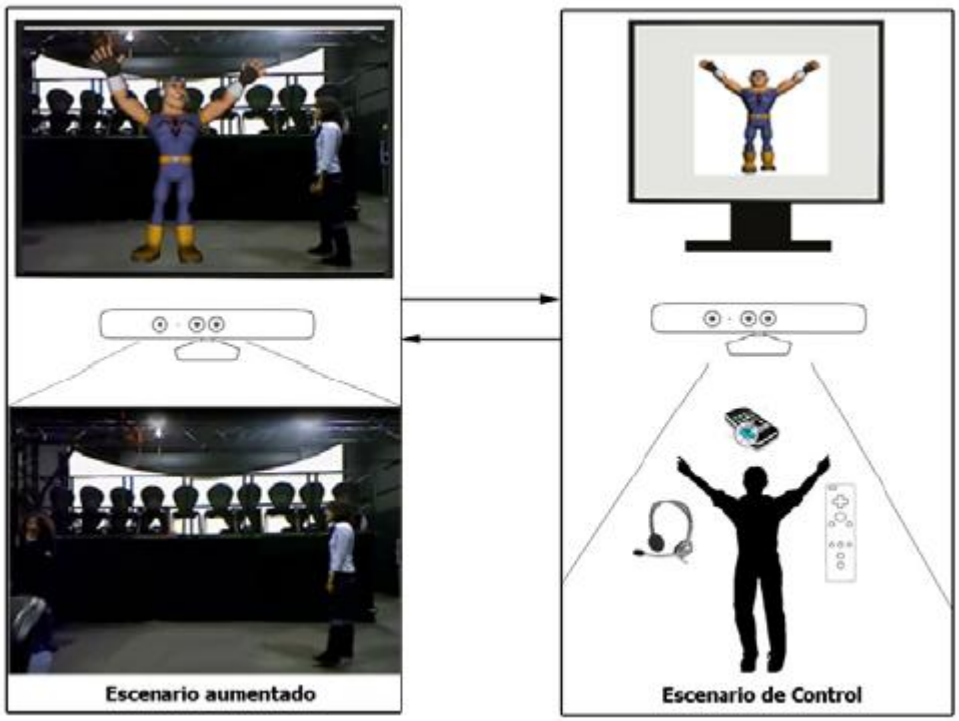


Figura 2. Estructura general del sistema [8].

La Universidad Autónoma de Barcelona utilizó el dispositivo Kinect como método de motion capture o mocap, es el proceso de grabar el movimiento de objetos o de personas. Para aplicarlo a animación 3D aplicada en video juegos.

El proyecto recolecta la información de los cálculos que hace NITE sobre el esqueleto y la puede traducir en formato BVH para otros programas de diseño, como el 3dsMax, este método guarda la información que se recolecta en el motion capture [9].

En el programa marco de la Unión Europea en asocio con el grupo Biomecánico de Valencia España se creó un sistema basado en tecnología de la información para ayudar a las personas mayores a prevenir caídas a través de contenidos educativos y programas de ejercicios con los que se evaluó su equilibrio. El sistema estará gestionado por una red social de información y conocimiento a la que el usuario tendrá acceso mediante su televisión y una consola de video juegos integrada.

El dispositivo principal del video juego es el sensor Kinect de Microsoft, que sirve para interactuar con la consola a través de los gestos, y al mismo tiempo monitorizar los movimientos que se realizan al ejecutar los juegos y ejercicios, [10] como lo indica la figura 3

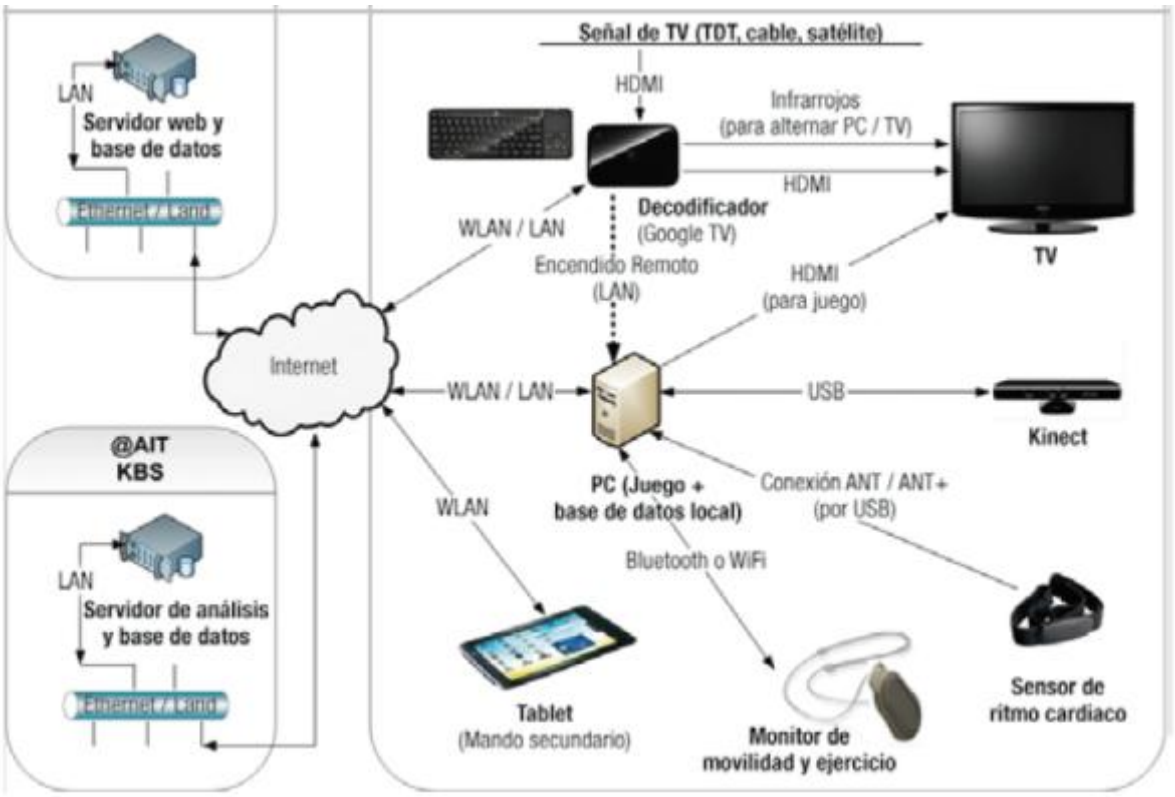


Figura 3. Arquitectura del sistema [11].

Utilizando una manera sencilla y natural pero sin dejar de ser compleja e interesante la Universidad de Madrid comenzó a trabajar en el mundo de la inteligencia ambiental, utilizando el sensor Kinect como interfaz de usuario, y partiendo de un entorno llamado AMILAB (Ambient Intelligence Laboratory) encargado de visualizar los gestos [12].

Así mismo en Catalunya se ha estudiado el desarrollo de aplicaciones para Windows controladas por Kinect utilizando software Development Kit (SDK, Kit de desarrollo de software). El cual se centra en la programación de video juegos aptos para su aplicación en terapias de neurohabilitación.

Es decir, en la habilitación de pacientes con dificultades motrices debido a daños cerebrales, cuyos problemas de motricidad son de nacimiento y no como consecuencia de algún accidente o causa que haya provocado la pérdida motriz. La neurohabilitación es un proceso medico complejo que ayuda a la recuperación de un daño en el sistema nervioso o compensar cualquier alteración funcional derivada del mismo. Utilizando como interfaz gráfica un XNA, que es una plataforma de desarrollo de video juegos para plataformas Windows. [13] Como se indica en la figura 4.

Figura 4. Manipulación de equipos [14].

Usando el software libre de Microsoft SDK y las imágenes RGB proporcionadas por la cámara de profundidad del Kinect se estableció la obtención de variables cinemáticas como: posición, velocidad y aceleración de determinados puntos de control del cuerpo de un individuo (como pueden ser la cabeza, cuello, hombros, codos, muñecas, caderas, rodillas y tobillos) para extraer patrones de movimiento. Cuyo objetivo es llegar a poblaciones con riesgo de exclusión (como es el espectro autista), en tele-diagnostico, y en general entornos donde se necesite estudiar hábitos y comportamientos a partir del movimiento humano. Sistema que se desarrolló en Madrid integrando el software libre de Microsoft aplicado al movimiento [15].

No solo se pueden tener patrones de movimiento sino que se pueden realizar una monitorización del cuerpo en 3D utilizando tecnología como Kinect, en visión por computador, las técnicas de seguimiento de movimientos en 3D se encuentran actualmente en un buen momento debido a tecnologías emergentes que permiten la captura de imágenes con información de profundidad. Dentro de estas tecnologías se encontró como destacada la cámara del Kinect, por su fácil acceso y su precio. Originalmente destinada solo al mundo de los video juegos, hoy día resuelve aplicaciones sobre PC que requieran básicamente seguimiento a usuarios [16], como lo indica la figura 5

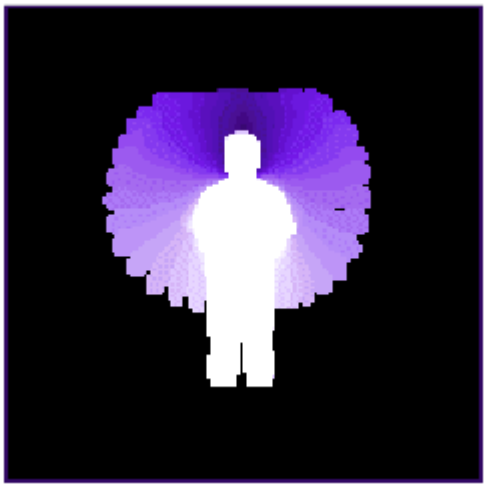


Figura 5. Historial de movimiento de un usuario usando tecnología Kinect [17].

En la Universidad de Zaragoza se realizó un estudio en dos campos de investigación, como son la robótica y la visión por computador. Tomando en particular el uso del dispositivo Kinect como interfaz para la interacción con el robot RoboNova-1.

Este robot dispone de un software propio para su programación desde Windows, la comunicación con el robot se realizara mediante un módulo diseñado para otro robot. Esta aplicación se divide en dos bloques: el primero contiene las aplicaciones que servirán para capturar los datos necesarios en el entrenamiento de los métodos de clasificación de gestos. El segundo bloque se encuentra la aplicación principal que hará uso de la interfaz gestual con la cual se evaluara el rendimiento de los métodos de clasificación [18].

La Universidad Carlos III de Madrid realizó un proyecto teniendo en cuenta robots o sistemas automáticos que pudieran moverse por entornos en los cuales puedan desenvolverse las personas. Con la finalidad de disponer de una movilidad integra que permita desempeñar trabajos e imitar las actividades cotidianas de los seres humanos, disponiendo de software y hardware que permita desarrollar dichas acciones.

Trabajo centrado en el diseño de software que, mediante la interpretación de imágenes tomadas con la cámara del Kinect, con el fin de estimar los planos por los que un robot o sistema equipado con dicho dispositivo pudiera transitar sin el riesgo de colisionar con ningún objeto. Mediante la toma de nubes de puntos, evaluar el punto que se encuentre dentro del plano formado por el suelo. Posteriormente obtener los puntos correspondientes a obstáculos posibles, proyectándolos y eliminándolos de la superficie del suelo para brindar un resultado valido y así acceder a zonas cruzables de estancias con objetos, pudiendo determinarlas en condiciones adversas y de poca visibilidad [19].

Pero no solo investigaciones, proyectos y análisis de problemáticas planteadas utilizando el sensor Kinect se da en España, también se encontraron trabajos utilizando la tecnología Kinect en Sur América, se encontró en Chile un proyecto enmarcado en el ámbito de la neo rehabilitación y la kinesiología. Utilizando la tecnología actual para detección de movimiento obteniendo resultados positivos que han aportado a la medicina y las ciencias de la salud.

Para la detección del movimiento se utilizaron varias tecnologías, una de ellas fue la video captura, empleando algoritmos de distinto tipo. También se le dio la utilización a acelerómetros, que detectan rotaciones en tres dimensiones, sensores infrarrojos y pantallas táctiles. Uniendo todas estas tecnologías para lograr una interacción humano-computador basada en movimientos corporales. Todo esto destinado a la rehabilitación y mejora de la calidad de vida de las personas que padecen alguna sintomatología asociada al movimiento, como lo indica la figura 6 [20].

Figura 6. Usuario en terapia de rehabilitación, haciendo uso de video captura [21].

En Argentina se le dio gran importancia y la disminución de los costos de los dispositivos, lo cual ha promovido la investigación y desarrollo de nuevas aplicaciones basadas en gestos. El dispositivo más popular es el Kinect, el cual permite el seguimiento de los movimientos de los usuarios. La aplicación presenta herramientas que facilitan el desarrollo de aplicaciones que interactúan por medio de gestos basados en Kinect.

Se presentan técnicas de aprendizaje de máquina que reconocen gestos a partir de las posiciones en el espacio de las partes del cuerpo provistas por el sensor, las evaluaciones mostraron que la precisión alcanza el 97% de los gestos evaluados y logra una reducción de un 67% del esfuerzo en términos de líneas de código [22].

En México se desarrolló una alternativa para representar y aprender gestos utilizando Modelos ocultos de Markov HMM y Kinect. Cuyo enfoque es utilizar la cámara del sensor para evitar que la extracción de los datos sea afectada por factores como la iluminación. Se usó la implementación de algoritmos de seguimiento de objetos y se guardaron los gestos (ordenes) del usuario para ser aprendidos.

Se procesaron los datos de los movimientos y se aproximaron utilizando HMM. Estos modelos son evaluados en un esquema de clasificación con 7 gestos diferentes [23].

La figura 7 muestra la forma en que son ejecutados cada uno de los gestos por la persona que está dando la orden, el gesto saludo es solo para introducir variedad en las distintas formas de los gestos y probar el poder de los HMMs para este tipo de curva oscilante.

Figura 7. Gestos ejecutados por el usuario [24].

En el Ecuador para proteger espacios y brindar seguridad tanto física como de información se desarrolló una investigación cuyo objetivo, es brindar una solución altamente fiable a los sistemas de seguridad de la central de monitoreo de data center de la empresa Asistencia PC. Para ello se utilizó el Kinect aprovechando sus diferentes sensores, que permita instalar un sistema con el objetivo de prevenir cualquier acceso no deseado a una consola de administración de Data Center.

Se implementará un prototipo que contará con un módulo de reconocimiento de gestos, que tendrá como entrada la percepción del mundo externo que será captada por los

sensores de equipo del Kinect, estos dispositivos procesaran dicha información para transformarla en una acción, para determinar la presencia del operador registrado. Cuando el operador abandona la consola de monitoreo y administración, el sistema se bloqueará automáticamente de manera inmediata la consola denegando el acceso a cualquier persona que no esté autorizada [25].

Así mismo en este mismo país se realizó un sistema de análisis de movimiento (SAM), proyecto de investigación que se desarrolló e implementó en el Grupo de Investigación en Ingeniería Biomédica (GIIB) de la Universidad Politécnica Salesiana. Formado por tres etapas: captura de movimiento, reconstrucción y animación tridimensional. Este sistema se basa en la tecnología de cámaras de profundidad que ofrece el dispositivo Kinect. Para el diseño e implementación se considera principalmente las características técnicas que presenta el sensor: los ángulos de incidencia, las distancias de reconocimiento, la velocidad de procesamiento y los ángulos de elevación del motor.

La etapa de captura de movimiento consta de un arreglo de dos dispositivos, este arreglo captura los movimientos mediante 20 puntos que representan las articulaciones que forman el esqueleto de usuario.

La etapa de reconstrucción tridimensional utiliza la librería Kinect SDK y la animación tridimensional utiliza la plataforma Blender, para que el modelo tridimensional imite los movimientos que realiza el usuario, se emplea la comunicación OSC. Donde, la interfaz principal adquiere los datos del usuario y los envía a la plataforma [26].

En la figura 8 se muestra la etapa de captura de movimiento formada por dos Kinect ubicados en el extremo del área de trabajo cuya vistas frontales están sobre el mismo eje y altura.

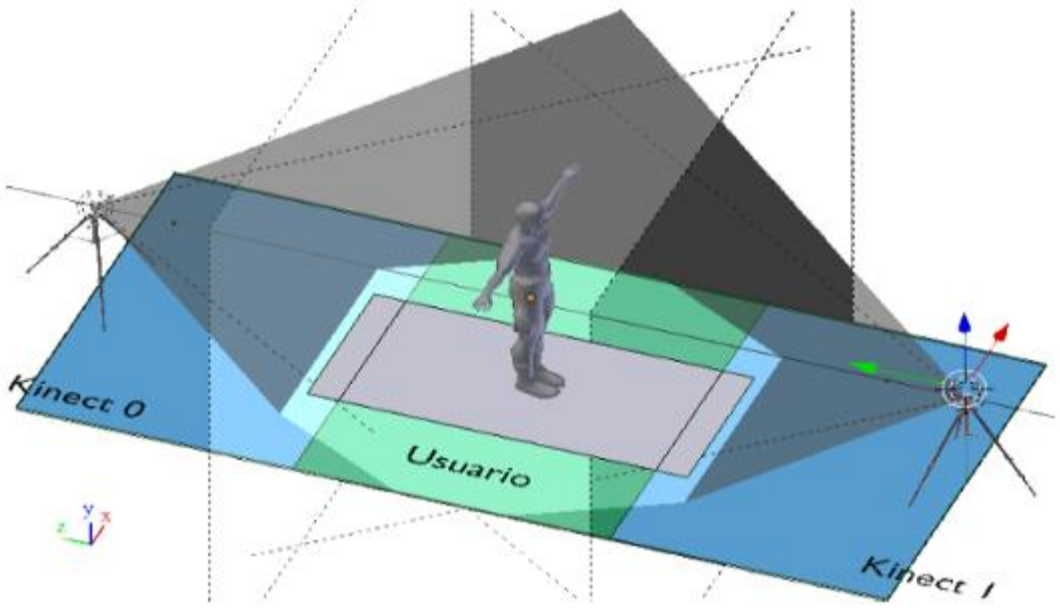


Figura 8. Perspectiva del espacio físico para la captura de movimiento [27].

En Colombia en la Universidad distrital Francisco José de Caldas, se realizó una investigación haciendo uso de la tecnología Kinect. Se desarrolló un dispositivo portátil, a partir de sensores de movimiento (acelerómetros), los cuales proporcionan las características cinemáticas de marcha humana tales como: velocidad, aceleración y ángulo formado entre los segmentos articulares (cadera, rodilla y tobillo).

El proyecto permite la captura de marcha humana en cualquier espacio no controlado, estableciendo parámetros que pueden ayudar a la medicina [28].

En la figura 9 se muestra la orientación de los acelerómetros en la extremidad izquierda y el acelerómetro de la extremidad derecha contrario para obtener una posición fija a cada articulación por parte de los sensores.

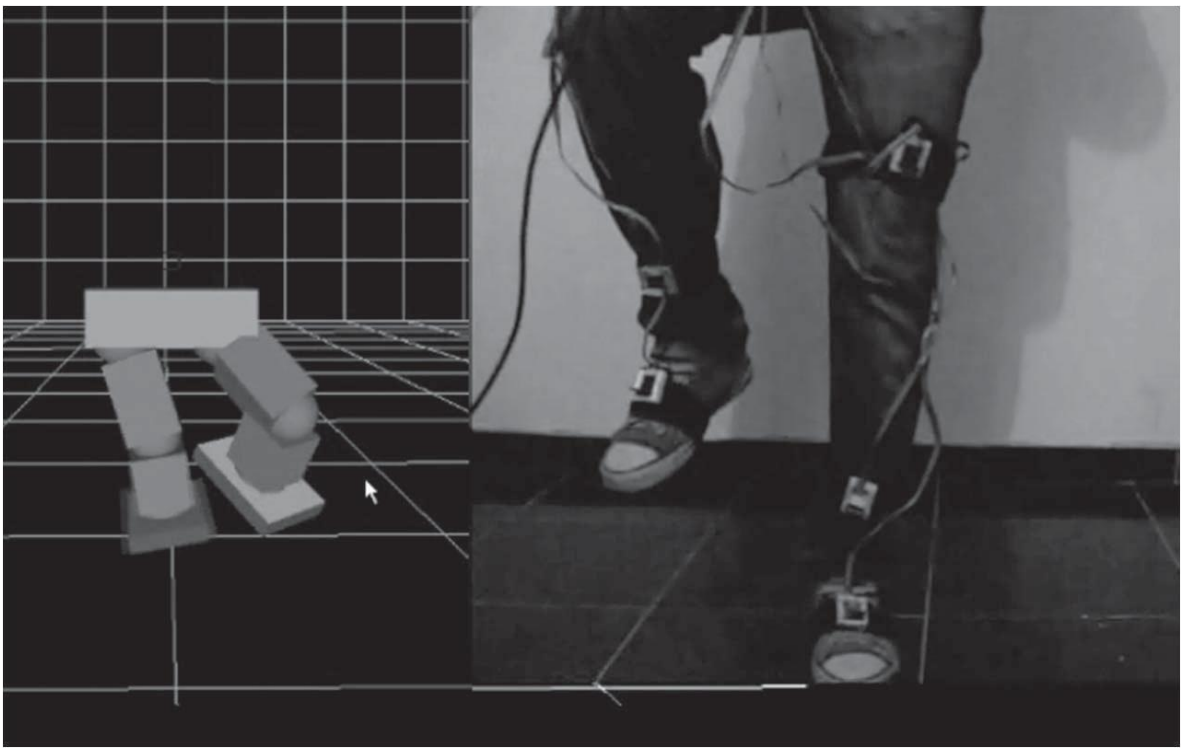


Figura 9. Captura de movimiento [29].

Así mismo, se le ha dado importancia a la línea de la robótica en nuestro país, se encontró un trabajo sobre un brazo robótico direccionado desde un PC por medio inalámbrico, investigación que se realizó en la Universidad del Quindío.

El robot empleado se encuentra dentro de la clasificación de robot móviles manipuladores. Para generar el direccionamiento del brazo se utiliza el Kinect como dispositivo de reconocimiento de movimientos; una vez realizado el procesamiento adecuado de la información captada, se transmiten los datos de forma inalámbrica utilizando tecnología

zigbee hacia el brazo robótico, compuesto por un sistema de recepción y un circuito microcontrolado que permite manipularlo [30].

En la Universidad Tecnológica de Pereira se encontró un artículo que muestra la creación de un novedoso sistema para la rehabilitación física de pacientes con múltiples patologías, a través de dinámicas de video juegos de ejercicios y el análisis de los movimientos de los pacientes usando un software desarrollado. Este sistema está basado en el uso del sensor Kinect para ambos fines: divertir al paciente en su terapia y proporcionarle al especialista una herramienta para el registro y análisis de datos de captura de movimiento (MoCap), tomados a través del sensor Kinect y procesados utilizando análisis biomecánico mediante la transformación angular de Euler [31].

En la figura 10 se puede observar el comportamiento del ángulo de precesión (frontal), el ángulo de nutación (sagital) y ángulo de rotación vs el tiempo de captura total en fotogramas.

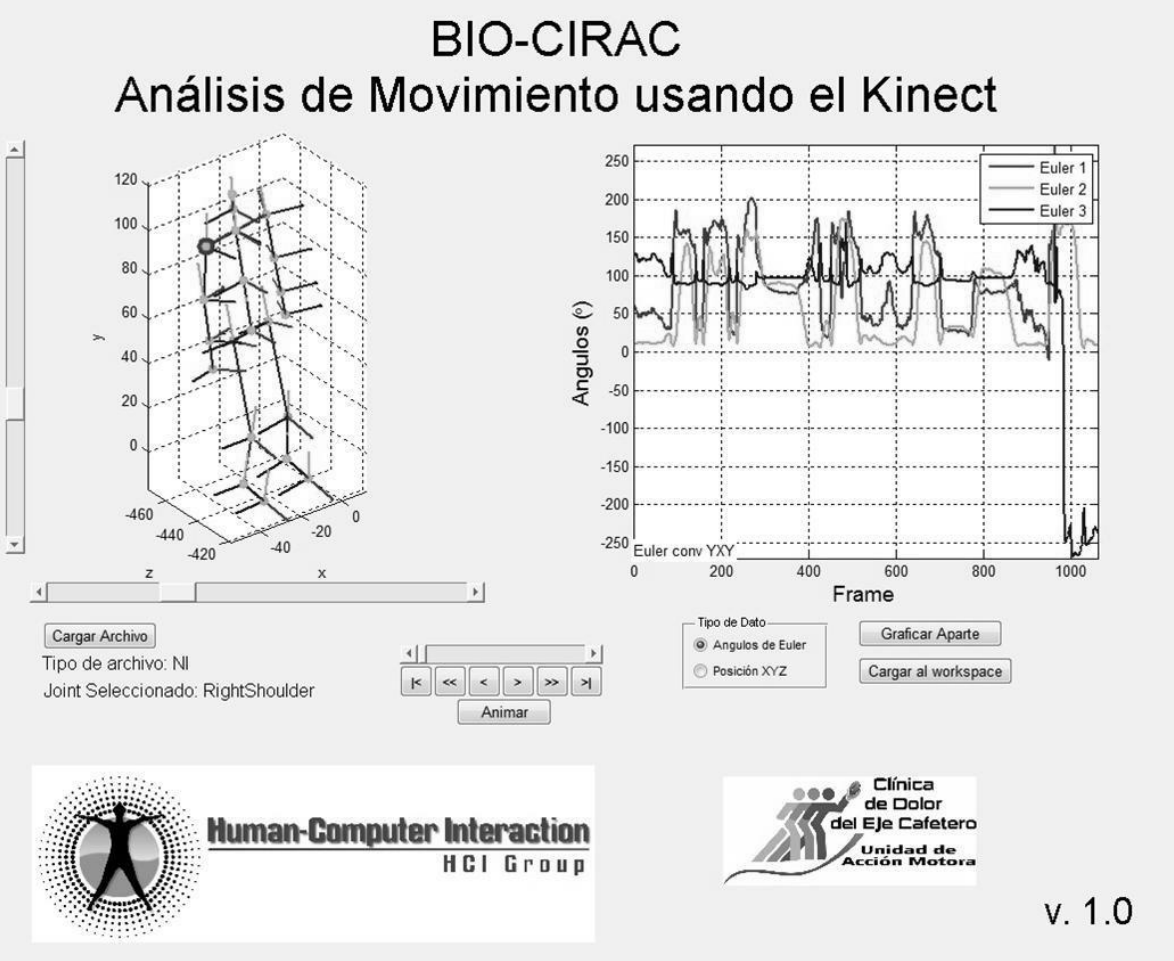


Figura 10. Archivo de captura de movimiento y sus graficas [32].

2. MARCO TEORICO

2.1 MICROSOFT KINECT

Kinect fue lanzado en noviembre de 2010 originalmente para la consola de video juegos Xbox 360 de Microsoft. A diferencia de los controladores de Sony y Nintendo, el dispositivo Kinect permite a los usuarios de la consola de video juegos Xbox 360 interactuar a través de su cuerpo o mediante su vox sin la necesidad de un control físico o un dispositivo táctil. Esto lo logra a través de la tecnología de una compañía Israelí llamada PrimeSense que permite obtener en tiempo real la profundidad, color y audio de una escena.

Desde su introducción al mercado es evidente que no sólo ha estado transformando la industria de los video juegos, sino también muchas otras áreas como robótica y realidad virtual. En la figura 8 se muestran los componentes que integran el dispositivo Kinect: una cámara RBG, un sensor de profundidad, un proyector láser de luz infrarroja, múltiples micrófonos y un motor sobre la base que permite modificar la inclinación del dispositivo.

Figura 11.Componentes del Kinect [33].

El sensor de profundidad, un sensor CMOS monocromático, funciona en conjunto con el proyector láser de luz infrarroja. Ambos permiten capturar imágenes incluso en completa oscuridad. A continuación se describen las especificaciones de los componentes de Kinect:

Visión horizontal de 57 grados

Visión vertical de 43 grados

Aproximadamente 31 grados de movimiento del motor (+ 31 hacia arriba, -31 hacia abajo)

Resolución de la cámara de profundidad de 640 X 480 píxeles a 30 fotogramas por segundo

Resolución de la cámara RGB de 640 X 480 píxeles a 30 fotogramas por segundo.

Consumo de energía de 2,25 watts. [34]

2.1.1 Arquitectura del Kinect

La estructura del sensor Kinect es similar a una cámara web. Incluye una cámara de profundidad, una cámara RGB y una matriz de 4 micrófonos, que aíslan las voces del ruido ambiental permitiendo utilizar al Kinect como un dispositivo para charlas y comandos de voz. Se encuentra sobre una base motorizada, controlada por un acelerómetro de 3 ejes, que le permite rotar horizontalmente para ajustar el campo de vista de las cámaras para ver el cuerpo completo del usuario.

En la figura 12 muestra un diagrama de los componentes de hardware del procesador de imagen de Kinect. Este es operado por el sistema en chip PS1080, desarrollado por Primesense, que se encarga de la generación y sincronización de las imágenes de profundidad e imágenes de color. El chip ejecuta todos los algoritmos de adquisición de imágenes de profundidad de la escena a partir del sistema de proyección de un patrón de puntos infrarrojos llamado LightCoding.

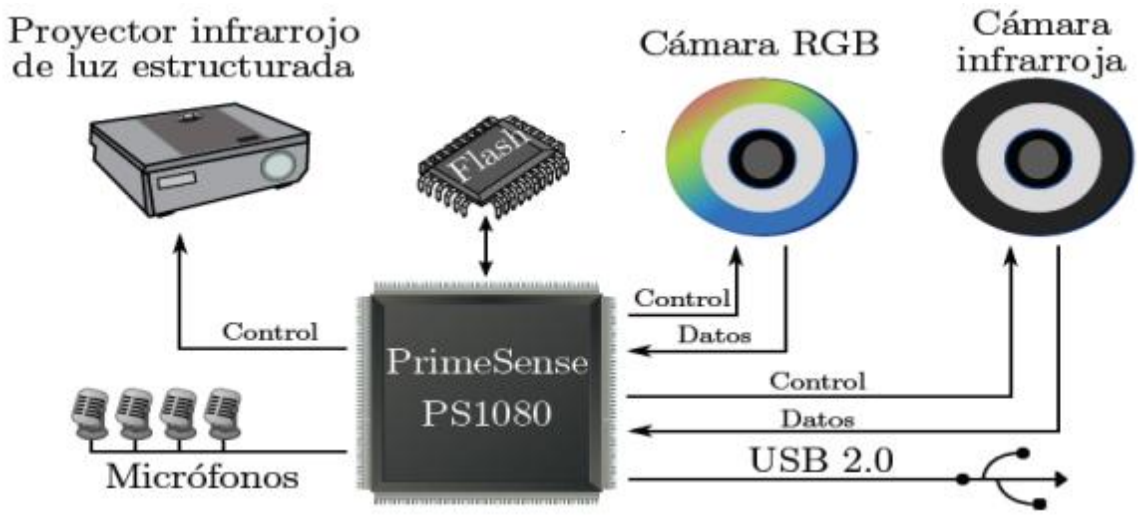


Figura 12. Diagrama del hardware de Kinect [35].

Kinect funciona bajo un esquema maestro-esclavo, donde el maestro es la computadora y el esclavo es el Kinect. El chip genera y mantiene en memoria los cuadros de imagen de profundidad y color a una velocidad de 30 cps (cuadros por segundo). El acceso a estos datos se realiza a través de un puerto USB 2.0 especial. El procesamiento del audio y el control de USB es realizado por un microprocesador Marvell Technology que funciona independiente al procesamiento de imágenes.

2.1.2 Cámara de profundidad. La cámara de profundidad se compone por la cámara infrarroja y el proyector infrarrojo de luz estructurada, como se muestra en la figura 13.

Figura 13. Cámaras de Kinect [36].

La cámara de profundidad utiliza una tecnología de codificación por luz llamada LightCoding desarrollada por PrimeSense que actúa como un escáner 3D para realizar una reconstrucción tridimensional de la escena. LightCoding es similar a los escaners de luz estructurada, pero en lugar de desplazar una línea de luz, se proyecta un patrón de puntos infrarrojos en la escena. El patrón de puntos se localiza en un difusor (rejilla) frente al proyector infrarrojo. Al emitir la luz infrarroja, el patrón se dispersa por la escena, proyectándose sobre las personas y objetos presentes en ésta.

Como se describe en la patente del sistema de PrimeSense, el patrón se mantiene constante a través del eje Z (“constante” se refiere a que la forma del patrón no cambia si se proyecta en una superficie perpendicular al eje focal de la cámara dentro del rango de especificación de 1.2 a 3.5 m), sin embargo, al existir objetos en la escena el patrón se

proyecta o adapta a la forma de éstos, con lo que se obtiene un desplazamiento del patrón en el plano XY de la imagen infrarroja en relación al patrón. Este desplazamiento es el que permite realizar una triangulación entre los puntos del patrón y los puntos del patrón proyectados en la imagen, para realizar la reconstrucción tridimensional de la escena y obtener la imagen de profundidad.

2.1.3 Reconstrucción de objetos. Kinect reconstruye la escena visualizada en 3D para obtener las imágenes de profundidad. La reconstrucción de objetos es un área especial de visión por computador. Se encarga del desarrollo e investigación de técnicas para reconstruir objetos tridimensionales y el cálculo de la distancia entre el sensor y los objetos de la escena. Es utilizada para identificar y conocer la distancia hacia objetos en la escena y detección de obstáculos (robótica y sistemas de control vehicular), inspección de superficies en sistemas de control de calidad y navegación de vehículos autónomos. También es utilizada en la estimación de objetos tridimensionales (sistemas de ensamblado automático). La reconstrucción de objetos involucra otras áreas como el procesamiento de imágenes (filtrado, restauración y mejora de imágenes, entre otras) y reconocimiento de patrones (segmentación, detección de bordes y extracción de características, entre otras).

Una de las primeras técnicas usadas para estimar la forma de un objeto o mapa 3D se basa en la triangulación [37].

2.1.4 Triangulación. Se refiere a el proceso para determinar la distancia de un punto en un espacio 3D a partir de la visión de dos o más imágenes. La figura 14 muestra la geometría epipolar de un par de cámaras de visión estéreo.

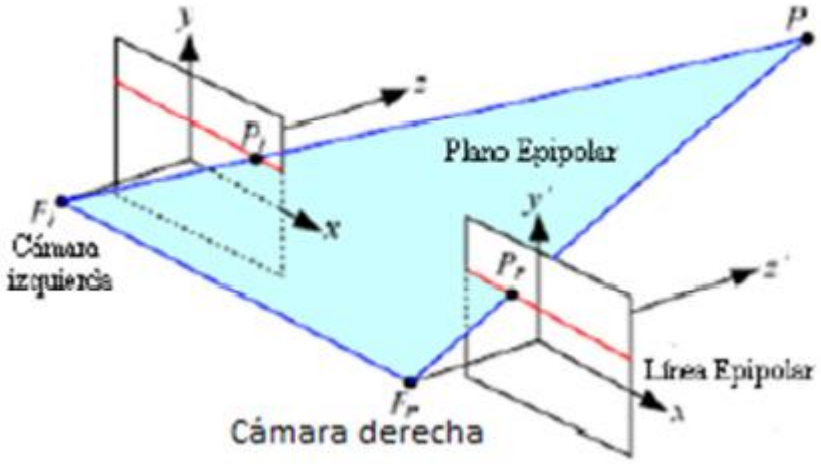


Figura 14. Triangulación en visión estero [38].

Un punto “P” es proyectado en un espacio tridimensional, que pasa por el punto focal de cada cámara, lo que resulta en dos puntos correspondientes de imagen, si estos puntos son conocidos y la geometría de la imagen es conocida, la proyección de esas dos líneas pueden ser determinadas. Usando algebra lineal se puede determinar la distancia de ese punto. Esta técnica es empleada por el sensor Kinect para calcular la profundidad de los objetos y obtener una reconstrucción 3D, la triangulación es realizada por cada punto entre la imagen virtual y el patrón de observación.

El proceso de triangulación para obtener la profundidad de los objetos es mostrado a continuación:

Patrones de luz son proyectados en la escena

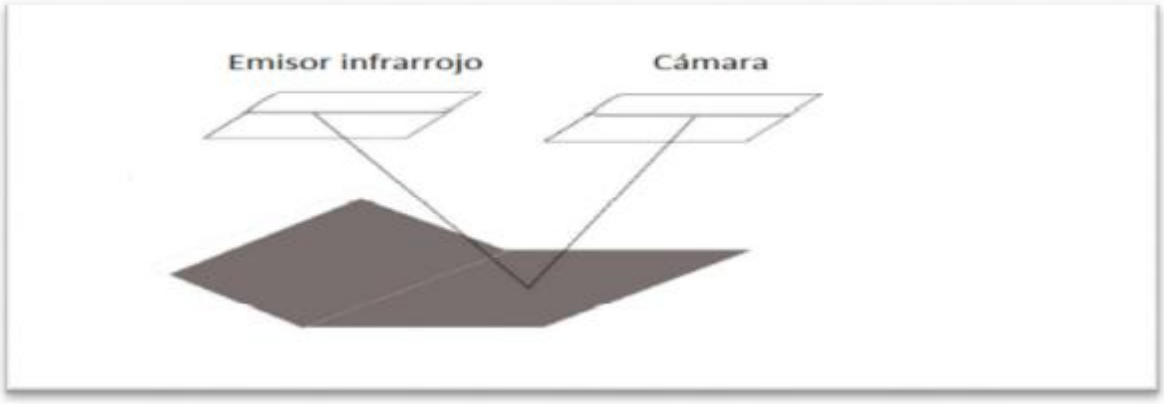


Figura 15. Imagen que muestra la proyección de puntos sobre una superficie [39].

Los patrones son distinguibles uno de otro

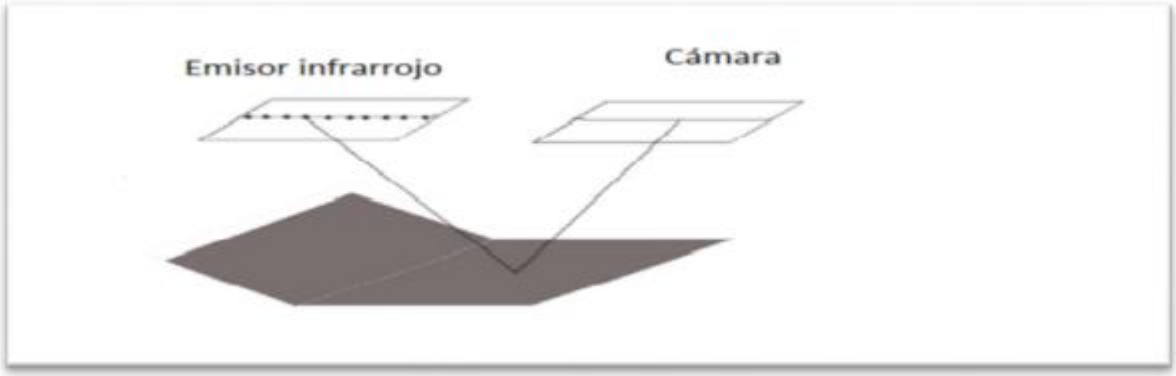


Figura 16. Proyección de puntos s [40].

Una imagen de profundidad es capturada para usarse como referencia

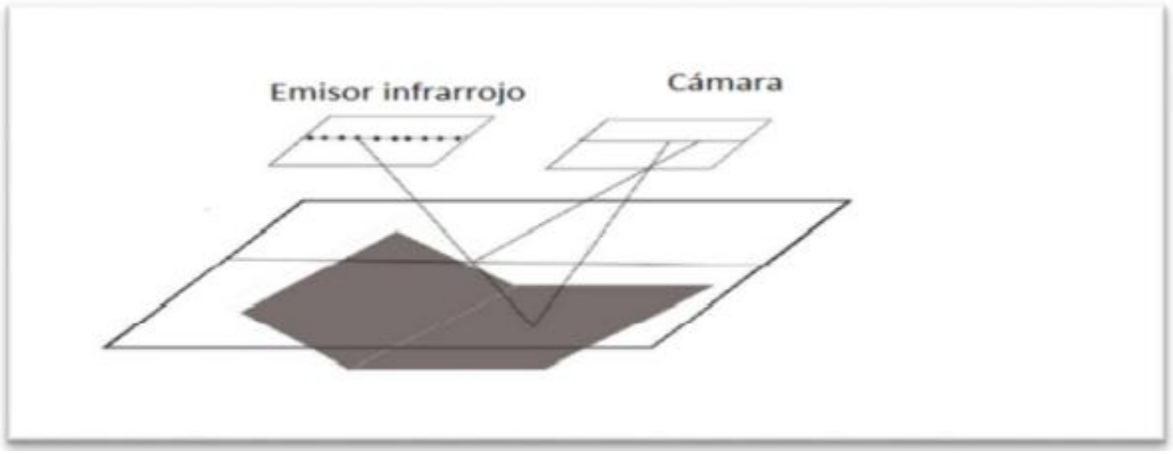


Figura 17. Imagen de profundidad capturada por la cámara [41].

Δx es proporcional para la profundidad del objeto

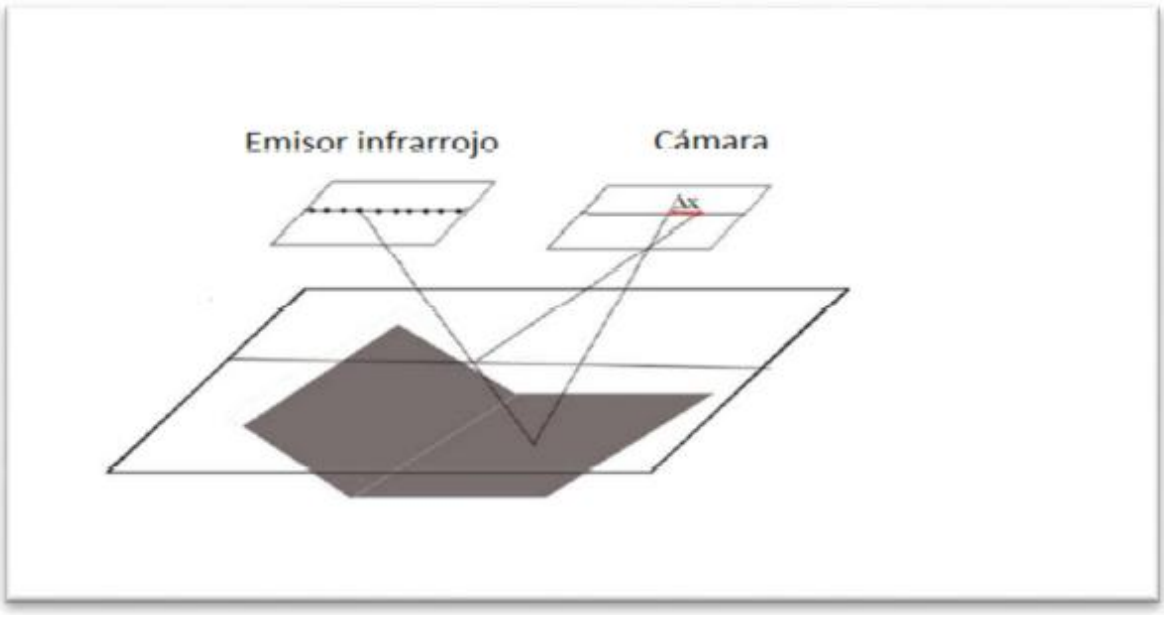


Figura 18. Δx para el punto proyectado [42].

2.1.5 Imágenes de profundidad. El sensor Kinect entrega imágenes de profundidad con una resolución de 640*480 píxeles. Los datos de profundidad son representados en un espacio de color RGB o YUV (dependiendo de la configuración), en 16 bits. Para el caso

del espacio RGB se utilizan 16 bits para establecer los datos de profundidad, los cuales corresponden al canal R(rojo) y al canal G(verde), el canal B(azul) se utiliza para establecer datos de usuario.

2.1.6 Detección y seguimiento de usuario. Cuando se adquieren datos a través de la cámara de profundidad, imagen con la información de la distancia de los objetos o usuario, esta imagen es analizada para extraer información que esté relacionada con el usuario, su posición y la pose realizada. El sensor contiene dentro de su software un algoritmo de detección que consiste en dividir el cuerpo del usuario en 31 partes que son reconocidas en un plano tridimensional. Una representación del proceso de detección de usuario se muestra en la figura 19.

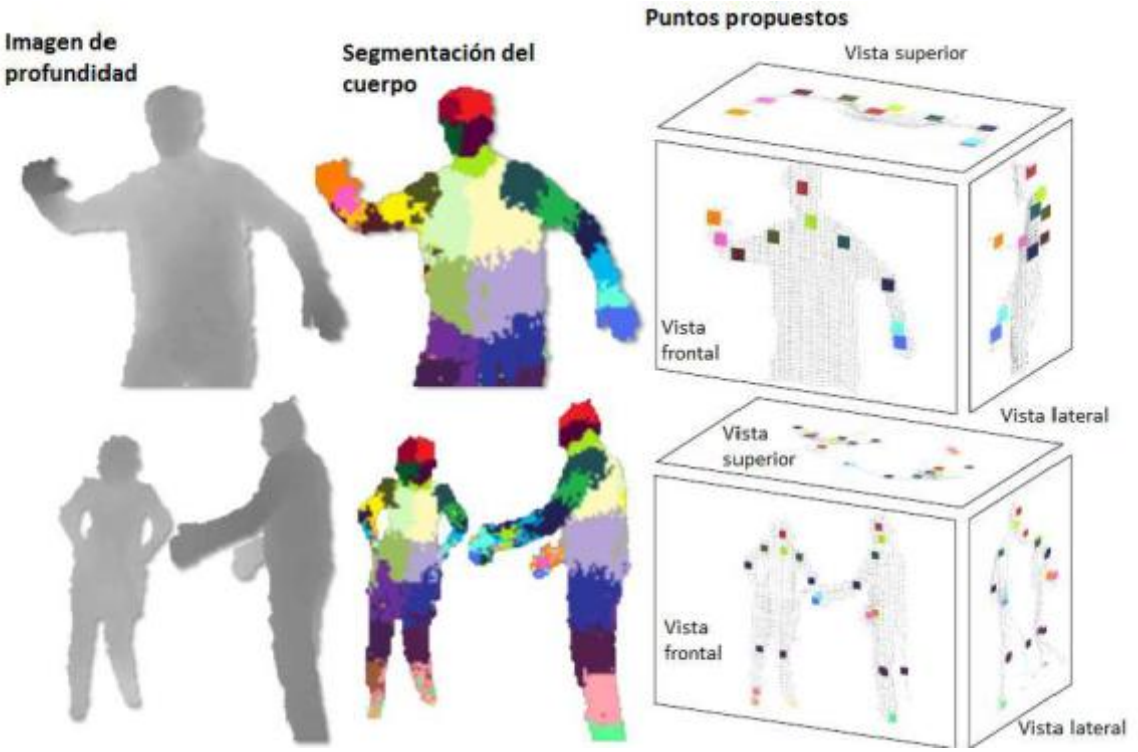


Figura 19. Proceso de segmentación y obtención de articulaciones a partir de imágenes de profundidad [43].

El análisis de imágenes de profundidad tiene ventajas sobre el análisis de imágenes a color, las imágenes de profundidad proporcionan mayor calidad en condiciones ambientales ligeras, es decir, en espacios en donde la luz solar no afecte de forma directa. Esto hace que el sensor realice muestras de dichas imágenes a través de la cámara de profundidad y estas sean más fáciles de analizar que las imágenes a color. Para segmentar la imagen del cuerpo del usuario en diferentes imágenes se utiliza en clasificador de imágenes, cuyo algoritmo está basado en un árbol de decisiones. Este algoritmo utiliza una base de datos con una similar pose para cada imagen, con

aproximadamente 100 000 poses para cada imagen. Para establecer la decisión de que imagen corresponde a la imagen del usuario es establecen diferentes características. Una característica está definida como una función f(I,x) que define una imagen I y una posición x, además de un parámetro θ que describe las características de la imagen dentro del espacio tridimensional, a través de estas características, el algoritmo detecta que imagen se asemeja a la imagen almacenada dentro de la base de datos. A partir de este algoritmo el sensor obtiene articulaciones o puntos específicos del cuerpo del usuario.

Para obtener las articulaciones del cuerpo del usuario se llevan a cabo tres pasos:

- Un estimador de densidad por cada parte del cuerpo del usuario, basado en la probabilidad de la superficie del área de cada pixel
- Técnica para encontrar eficiencia en la densidad
- Algoritmo que obtiene la aproximada localización de los puntos o articulaciones

La detección de articulaciones es demasiado precisa, con un 91,4% de probabilidad de ser localizados correctamente dentro del plano tridimensional.

Las articulaciones o puntos del cuerpo, de los cuales se puede realizar un seguimiento dentro del plano que abarca el sensor, son los siguientes, como se muestra en la figura 20:

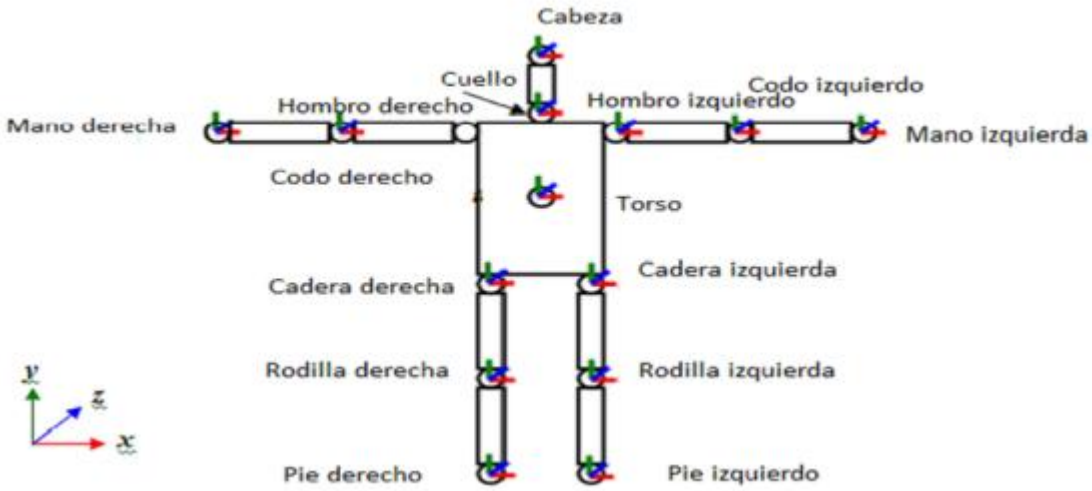


Figura 20. Articulaciones del cuerpo. [44].

2.1.7 Rastreo del esqueleto. Consta de procesar las imágenes de profundidad obtenidas con Kinect para detectar formas humanas e identificar las partes del cuerpo del usuario, presente en la imagen. Cada parte del cuerpo es abstraída como una coordenada 3D o articulación. Un conjunto de articulaciones forman un esqueleto virtual para cada imagen

de profundidad de Kinect, es decir, se obtienen 30 esqueletos por segundo. Las articulaciones generadas varian de acuerdo a la biblioteca de Kinect que se utilice. En OpenNI/NITE, cada esqueleto como lo indica la figura 18 está formado por 15 articulaciones ai = {xi, yi, zi} con zi > 0 cuyas coordenadas se encuentran expresadas en milímetros con respecto a la posición de Kinect en la escena. En el SDK de Microsoft y en la consola Xbox se añaden 5 articulaciones (los tobillos, las muñecas y el centro de la cadera).

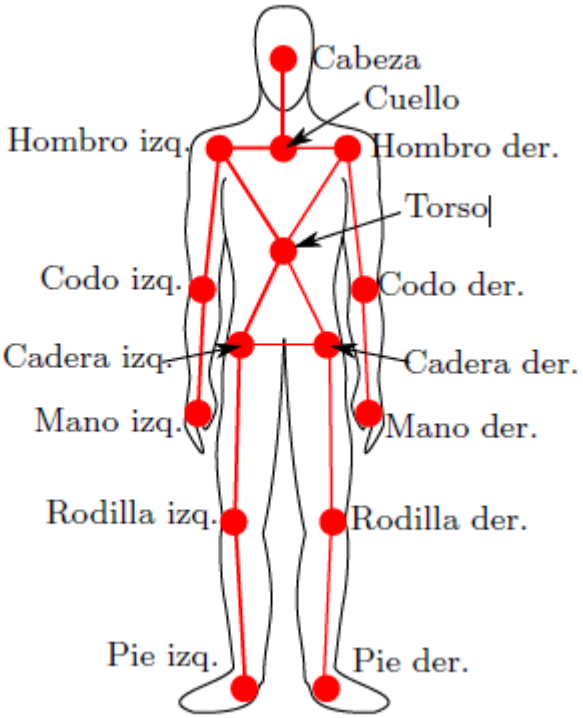


Figura 21. Esqueleto virtual de OpenNI/NITE [45].

El rastreo del esqueleto de OpenNI/NITE se inicia al realizar la pose de calibración “psi” levantando los brazos durante dos segundos como se muestra en la figura 22.

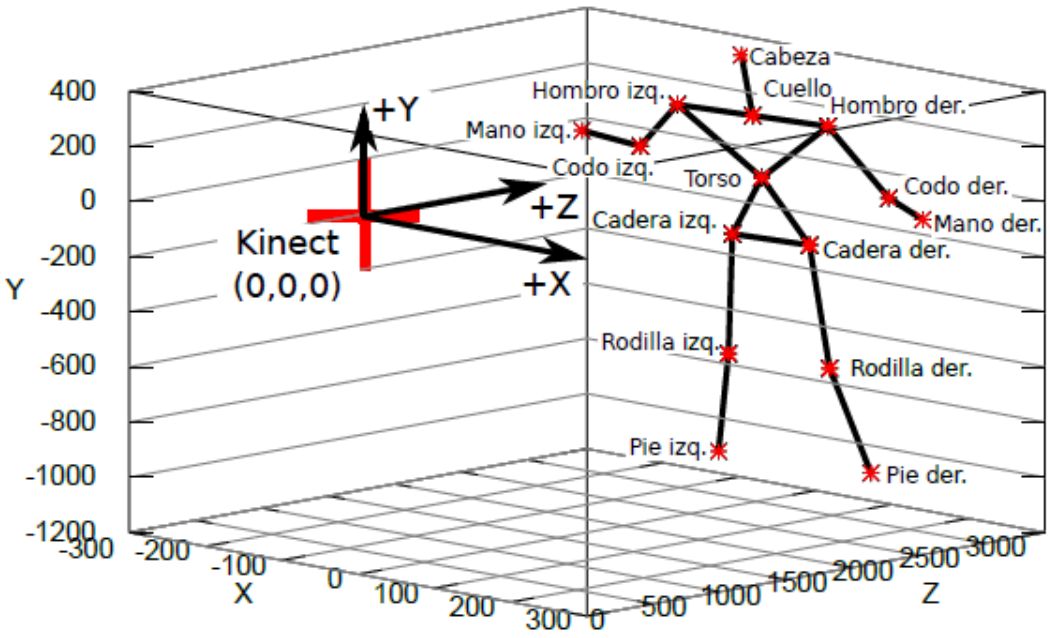


Figura 22. Configuración de la escena del esqueleto en OpenNI/NITE. Las articulaciones se encuentran en milímetros con respecto al origen del sistema de coordenadas (posición del Kinect) y en espejo (el eje X está invertido para el lado izquierdo del esqueleto corresponda al lado derecho del cuerpo y viceversa). Kinect mira hacia el lado positivo del eje Z. [46].

Para segmentar cada parte del cuerpo los autores construyeron un clasificador de árboles de decisión aleatorios o bosque aleatorio de árboles y un conjunto de más de 500,000 imágenes de profundidad de secuencias de humanos (reales y generadas por computador) de diversas complexiones y estaturas realizando acciones como correr, saltar, nadar y apuntar, entre otras. El clasificador se entrenó con un subconjunto de 100,000 imágenes de profundidad de la base de datos en donde las poses difieren en más de 5 cm. El clasificador no usa información temporal es decir busca poses estáticas en cada imagen de entrada y no movimientos entre imágenes de entrada consecutivas.

El clasificador de bosque es un conjunto de T arboles de decisión, donde cada nodo hoja almacena la distribución ci aprendida, que etiqueta a la parte Cí del cuerpo y los nodos rama contienen en una característica ғ y un umbral т.

La técnica de rastreo del esqueleto inicia tomando una sola imagen de profundidad de Kinect (de 640 X 480 pixeles). Para clasificar un pixel x de la imagen de entrada, se comienza en la raíz de un árbol del clasificador y se evalúa repetitivamente según la ecuación 1 (que permite saber si un pixel x está cerca o lejos de Kinect) para seguir la

rama izquierda o derecha de acuerdo al umbral τ en el nodo. Si la ruta conduce a un nodo hoja, se sabe que el pixel x pertenece a la parte Cí del cuerpo.

Para cada pixel x de la imagen de entrada, la característica f se calcula con

$$f(I, \boldsymbol{x}) = d_I\left(\boldsymbol{x} + \frac{\boldsymbol{u}}{d_I(\boldsymbol{x})}\right) - d_I\left(\boldsymbol{x} + \frac{\boldsymbol{v}}{d_I(\boldsymbol{x})}\right)$$

Ecuación 1. Característica f para cada pixel x de la imagen de entrada

Donde dI(x) es la profundidad del pixel x y los parámetros (u, v) son desplazamientos en coordenadas globales que al normalizarse por $\frac{1}{dI\,(x)}$ permiten que la característica sea invariante a la profundidad y a la traslación en el espacio tridimensional.

Finalmente, cada articulación del esqueleto se encuentra a partir del cálculo de la media de cambio (mena-shift) de los pixeles que forman la parte del cuerpo ci que se relaciona a la articulación de interés [47].

2.2 BIBLIOTECAS LIBRES DE KINECT

Debido al crecimiento en los últimos años en el área de procesamiento de imágenes y visión por computador, han surgido bibliotecas de desarrollo gratuitas que permiten, no solo realizar nuevos entornos sino que se ajustan a diferente software para realizar nuevos desarrollos encaminados a diferentes líneas.

2.2.1 Processing. Es la herramienta ideal para crear gráficos a tiempo real mediante programación. Es software libre por lo que cuenta con novedosas funciones creadas por usuarios alrededor del mundo

Processig también trabaja con librerías por lo que facilita el proceso de programación y es fácilmente adaptable a las situaciones que el usuario desee. Creada por Daniel Shiffman. Esta librería permitió la comunicación de Processing con el sensor haciendo más fácil y endebles los valores que devuelve el sensor [48].

2.2.2 OpenKinect. Es una librería que se ha venido desarrollando mediante técnicas de ingeniería inversa del protocolo utilizado por el Kinect por parte de una gran comunidad de desarrolladores. Esta librería es de código abierto y ofrece control sobre la mayoría de los

dispositivos de hardware del sensor, más no ofrece facilidades en cuanto a algoritmos especializados para el procesamiento de imágenes.

2.2.3 SDK. Se trata de un framework desarrollado por la empresa PrimeSense que ofrece gran facilidad para el desarrollo de aplicaciones utilizando el sensor Kinect. Al igual que OpenKinect ofrece control sobre la mayoría de los dispositivos de hardware del Kinect, además, posee compatibilidad con la librería NITE de PrimeSense se contiene algoritmos para el procesamiento de imágenes, detección y seguimiento de individuos. Sin embargo, el código de esta librería no es abierto.

Al ver que otras empresas generaban controladores y que tenía éxito, provoco que Microsoft decidiese publicar la SDK oficial de Kinect.

El SDK está orientado a la investigación académica principalmente aunque también a programadores particulares con el objetivo que experimenten con la creación de interfaces naturales de usuario.

El SDK permite:

- Skeletal Tracking de uno o dos personas que estén en el ángulo de visión de Kinect
- Cámara de profundidad que será capaz de calcular la distancia de los objetos al sensor de Kinect
- Procesamiento de audio para sus cuatro micrófonos

El SDK incluye:

- Drivers para usar Kinect con Windows
- API´s, interfaces de los dispositivos con documentación para desarrolladores
- Ejemplos de código de fuente, como lo indica la figura 23

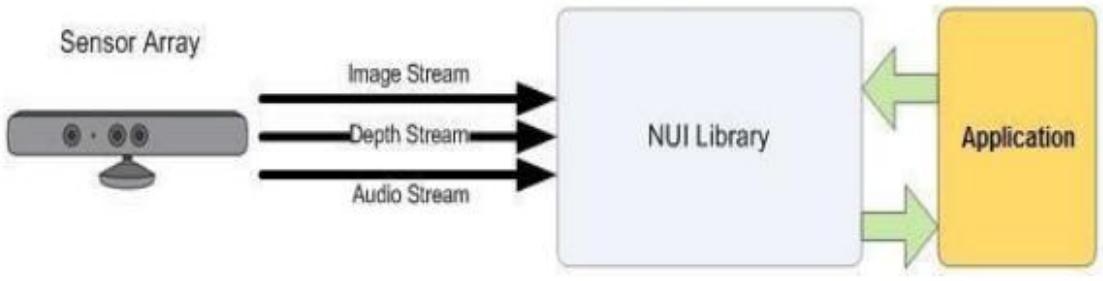


Figura 23. Interacción de software y hardware con la aplicación [49].

Los lenguajes utilizados por el SDK son C#, C++, Matlab y Labview. [50]

2.2.4 OpenNI. Es una organización sin ánimo de lucro formada para certificar y promover la compatibilidad e interoperabilidad entre dispositivos y aplicaciones que utilizan un paradigma de la interacción natural (NI, Natural Interaction).

En el ámbito de la informática, una interfaz natural es el término utilizado por diseñadores y desarrolladores de interfaces humano-máquina para referirse a aquellas que son efectivamente invisibles para el usuario y están basadas en la naturaleza o elementos naturales.

El término “interfaz natural “se usa en contraposición a las interfaces habituales, que requieren el uso de dispositivos externos artificiales cuya forma de funcionamiento ha de ser aprendida. Una interfaz de usuario natural (NUI) se basa en la capacidad del usuario de pasar rápidamente de novato a experto en su manejo.

Aunque no forma parte del hardware ni de sus usos originales. Kinect es un dispositivo cuyas características lo hacen propio para ser utilizado en el diseño de interfaces de gestos, que se conocen como Kinect User Interfaces (KUI, interfaces de usuario), las cuales forman parte de las interfaces naturales.

Dándose cuenta del enorme potencial del dispositivo de Microsoft en el desarrollo de las interfaces naturales, OpenNI, junto con PrimeSense, la compañía desarrolladora del hardware, y Willow Garage, un grupo de desarrolladores de software Open Source para robótica, crean un driver de código abierto para la utilización de Kinect en PC.

La estructura de trabajo (o Framework) OpenNI permite, por un lado, comunicarse con los sensores de audio, video y sensor de profundidad de Kinect, mientras que proporcionan una API que sirve de puente entre el hardware del equipo y las aplicaciones e interfaces del S.O. La idea es facilitar el desarrollo de aplicaciones que funcionen con interacción natural, gestos y movimientos corporales.

Actualmente OpenNI permite la captura de movimiento en tiempo real, el reconocimiento de gestos con las manos, el uso de comandos de voz y utiliza un analizador de escena que detecta y distingue las figuras en primer plano [51].

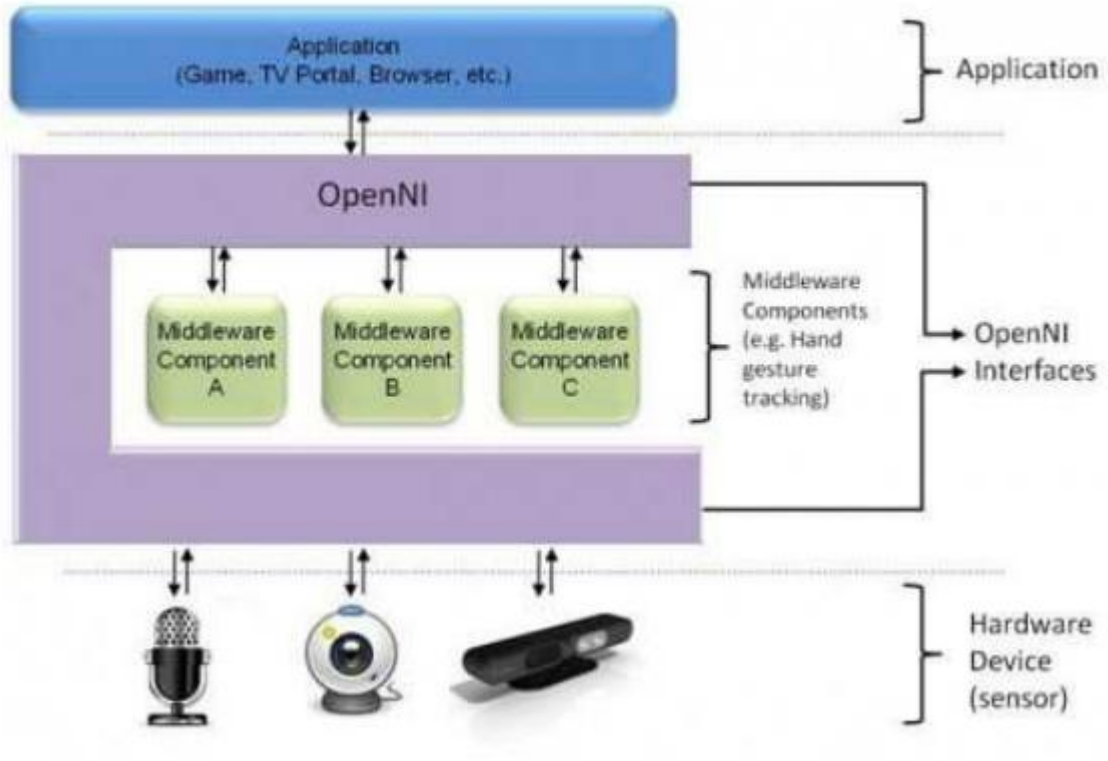


Figura 24. Estructura del framework OpenNI [52].

3. CAPTURA DE MOVIMIENTO

3.1 IMAGEN DIGITAL

Es la imagen utilizada por el computador, ésta se presenta digitalizada en forma de una matriz con una resolución de MXN elementos, cada elemento de la matriz se llama píxel, tiene un valor asignado que corresponde al nivel de luminosidad del punto de la escena captada.

En imágenes en blanco y negro, se almacena un valor por cada píxel, este valor corresponde al nivel de intensidad o nivel gris. El rango de valores utilizados para su representación varía entre 0 y 255, donde 0 representa el negro absoluto y 255 el blanco absoluto.

En las imágenes a color, los elementos de la matriz están compuestos por tres elementos, los cuales representan cada uno de los componentes básicos del color. Estos componentes son: Rojo (R), Verde (G) y Azul (B). En este caso el (0, 0, 0) representa el negro absoluto y (255, 255, 255) el blanco absoluto; la combinación de distintos valores ofrecen otros colores.

3.2 IMÁGENES A COLOR

Cuando se habla de color real se está hablando de imágenes capturadas por medio de sensores de color, cámaras o escáner.

Si una luz es acromática es decir, sin color esto indica que su único atributo es la intensidad o cantidad de luz.

Si una luz es cromática, se emplean tres magnitudes básicas:

- Radiancia: cantidad total de energía que sale de la fuente luminosa; se mide en watios (W).
- Luminancia: Medida en lúmenes (lm), proporciona una medida de la cantidad de energía que un observador percibe de una fuente luminosa.
- Brillo: es difícil de medir. Es uno de los factores fundamentales para describir las sensaciones del color. Es capaz de alterar la gama tonal ya que reduce el contraste de la imagen y la pérdida de detalle, según el nivel que se aplique

Los colores primarios se pueden sumar para obtener los colores secundarios. ; mezclando los colores primarios o uno secundario con su color opuesto primario en proporciones adecuadas se obtiene luz blanca.

Colores primarios de luz: rojo, azul, verde.

3.2.1 Modelos de color. Un modelo de color es la especificación de un sistema de coordenadas tridimensionales, en donde cada color es representado por un único punto. Los modelos de color más utilizados para el procesamiento de imágenes son:

- El modelo de color RGB (Red, Green, Blue). En este modelo los componentes son los colores rojo, verde y azul. Este modelo está basado en un sistema de coordenadas cartesianas. Las imágenes del modelo de color RGB consisten en tres planos de imagen independiente. La mayoría de cámaras a color utilizan este formato.
- El modelo de color CMY (Cian, Magenta, Yellow). Es utilizado por la mayoría de dispositivos que depositan pigmentos coloreados sobre el papel, como son las impresoras y fotocopiadoras
- El modelo de color CMYK (Cian, Magenta, Yellow, Black). Es un modelo de color basado en la síntesis sustractiva, según la cual, la mezcla a partes iguales de tres primarios (cian, magenta y amarillo), en su máxima intensidad, resulta en negro. Si se mezclan los colores primarios, se obtiene los colores secundarios [53].

3.3 CAPTURA DE MOVIMIENTO

El empleo de los sistemas de captura de movimiento empezó a finales de los años 70. Comenzó como una herramienta del análisis de investigaciones biomecánicas, donde se realiza un estudio del modo del caminar del ser humano y se investigan las fuerzas deformantes que sufre el cuerpo en un accidente; pero con el tiempo sus aplicaciones se han incrementado.

Figura 25. Análisis Biomecánico [54].

La captura de movimiento es la grabación de cualquier movimiento, bien sea de una persona o de un objeto, para su posterior análisis.

Los movimientos capturados pueden ser tan simples como la ubicación de un objeto en el espacio o tan complejos como el movimiento de la cara o de los músculos.

La captura de movimiento se utiliza para copiar los movimientos de un objeto real y pasarlos a un objeto creado por computador; es útil para crear animaciones con movimientos más reales que los creados por una animación manual [55].

El estudio del movimiento humano puede ser descrito como una ciencia interdisciplinar que describe, analiza y evalúa el movimiento humano. Existen disciplinas que tradicionalmente han tenido interés por el movimiento humano, como la Biomecánica, que tiene como objeto el desarrollo de los modelos del cuerpo humano que expliquen cómo se comporta éste mecánicamente y cómo se puede incrementar el rendimiento o disminuir las probabilidades de lesiones o fracturas del mismo.

Una técnica muy extendida para la obtención de las coordenadas de marcadores está basada en el análisis de imágenes y se denomina fotometría 3D (video o cine). Los datos 3D procedentes de imágenes suelen obtenerse manualmente, o bien de una manera invasiva (colocando marcadores en el cuerpo de una persona). La digitalización manual es la más comúnmente utilizada.

La investigación en la captura de movimiento humano está orientada actualmente hacia la implementación de un sistema general de seguimiento del cuerpo humano completo,

suficiente para manipular aplicaciones realistas, concentrándose en el estudio del movimiento articulado basado en modelos que no requieran marcadores externos.

3.3.1 El modelo de barras de Chen y Lee. Contiene 17 segmentos y 14 articulaciones que representan las características de la cabeza, torso, cadera, brazos, piernas y los parámetros del modelo son las coordenadas 3D de las articulaciones y la longitud de cada segmento rígido [56].

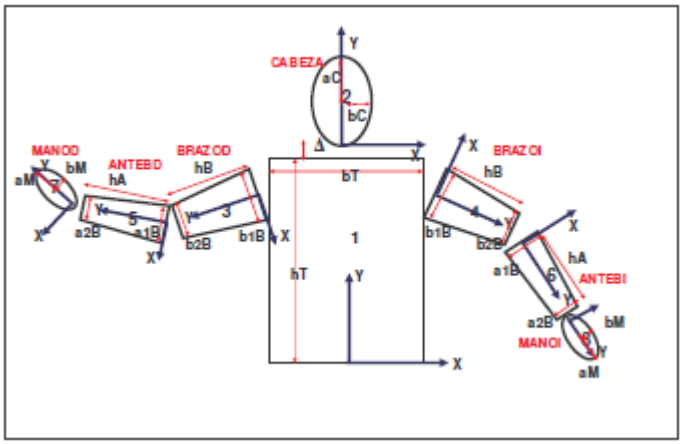


Figura 26. Modelo geométrico del hemisferio superior del cuerpo humano [57].

3.3.2 Captura de movimientos electromecánica. Es aquella técnica en la cual la captura de movimiento se realiza usando sensores mecánicos. En el proceso de captura de movimientos, la persona viste unos trajes especiales, adaptables al cuerpo humano, estos trajes son generalmente estructuras rígidas compuestas de barras metálicas o plásticas unidas mediante potenciómetros colocados en las principales articulaciones. Básicamente el actor coloca la estructura en su cuerpo y mientras se mueve, el traje se adapta a los movimientos que éste realiza, los potenciómetros recogen toda la información del grado de abertura de las articulaciones. El problema que tienen los sistemas de captura de movimiento electromecánicos con respecto a los otros sistemas es la incapacidad de medir translaciones globales (miden las posiciones relativas de los miembros, pero no el desplazamiento del actor en el escenario, es común el uso de sensores auxiliares para cubrir esta falencia de este tipo de sistema de captura). Por otro lado este sistema sume que la mayoría de los huesos humanos están unidos por articulaciones “bisagra”, pero no tienen en cuenta rotaciones complejas que se producen frecuentemente en las articulaciones humanas, como por ejemplo la que se presenta en los hombros o los antebrazos, las estructuras suelen ser pesadas y probablemente restringen el movimiento del actor, la figura 27 hace referencia al uso del traje.

Figura 27. Traje utilizado en captura de movimiento electromecánico [58].

3.3.3 Captura de movimiento electromagnética. Se dispone de una colección de sensores electromagnéticos que miden la relación espacial con un transmisor cercano. Los sensores se colocan en el cuerpo y se conectan a una unidad electrónica central, están constituidos por tres espiras ortogonales que miden el flujo magnético, determinando posición y orientación del sensor. Un transmisor genera un campo electromagnético de baja frecuencia que los receptores detectan y transmiten a la unidad electrónica de control, después se envía a un PC central, donde se infiere la información recopilada. Un sistema magnético típico consta de un transmisor, hasta 18 sensores, una unidad de control electrónica y un software propietario para procesamiento. El proceso de captura completo no es en tiempo real, pero se le aproxima bastante, dependiendo del tratamiento y procesamiento de la señal, así como del tiempo de respuesta de la conexión entre la unidad de control y el PC.

3.3.4 Captura óptica de movimiento. Los sistemas ópticos utilizan los datos recogidos por sensores de imagen para inferir la posición de un elemento en el espacio, utilizando una o más cámaras sincronizadas para proporcionar proyecciones simultaneas. Generalmente se usan indicadores (markers) pegados al actor, pero los sistemas más recientes permiten recoger datos fiables rastreando superficies del sujeto identificadas dinámicamente.

Estos sistemas producen datos con 3 grados de libertad para cada indicador; la orientación de una superficie se infiere utilizando la posición relativa de al menos 3 indicadores. Los sistemas ópticos de captura de movimiento son, en general, métodos muy fiables para capturar determinados movimientos cuando se utilizan sistemas de última generación. Además, permiten la grabación en tiempo real, con ciertas limitaciones como el número de indicadores, el número de actores y cámaras. Este tipo de sistemas pueden capturar un gran número de marcadores a frecuencias del orden de hasta 2000 fotogramas/ segundo, si bien hay que compensar la velocidad de captura con la

resolución. Existen sistemas de 4 megapíxeles a 360 hercios por 100000 dólares y sistemas de 0,3 megapíxeles a 120 hercios por 50000 dólares.

Figura 28. Captura de movimiento de la cara usando indicadores pasivos [59].

3.3.5 Mediante indicadores activos. En este sistema de captura, los indicadores emiten su propia luz (LED), con lo cual se consigue aumentar la distancia a la que se puede desplazar el sujeto. La posición de los indicadores se determina iluminando un indicador en cada instante de tiempo a una frecuencia muy alta (lo cual perjudicaría la frecuencia de muestreo), o bien varios indicadores a la vez, con procesamiento adicional para calcular la identidad de cada indicador a partir de su posición relativa. Para ello, los indicadores han de estar sincronizados con todas las cámaras para iluminarse en una sola captura.

Figura 29. Traje especial con indicadores activos [60].

3.3.6 Mediante indicadores activos modulados en el tiempo. Se trata de una mejora con respecto a los indicadores activos, en la cual los marcadores no se iluminan uno de cada vez, sino que se iluminan muchos a la vez mediante luz estroboscópica, determinándose la identidad de cada indicador mediante la frecuencia de destello. De esta forma se consiguen frecuencias de capturas mayores que con los sistemas activos estándar, con el inconveniente de aumentar la carga computacional. Los sistemas con indicadores activos modulados en el tiempo permiten aplicar los movimientos del actor sobre el personaje animado, permitiendo observar el resultado en tiempo real. Al existir un único identificador para cada indicador, se elimina el problema del intercambio de indicadores. Además, este sistema permite rodar al aire libre bajo la luz directa del sol. El procesamiento de los indicadores modulados se realiza en tiempo real en las cámaras, lo que permite tanto altas frecuencias de captura como mayor precisión a la hora de determinar la posición exacta del indicador, utilizando un algoritmo para aumentar la posición de la información capturada.

3.3.6.1 Sin marcadores. Son sistemas en los que el seguimiento de los movimientos de los actores no requiere que éstos vistan equipos especiales, utilizan algoritmos que analizan distintas fuentes de entrada de imágenes identificando formas humanas y descomprimiéndolas en trozos para realizar el seguimiento de sus movimientos. Estos sistemas trabajan bien con movimientos generales pero suelen tener dificultades con movimientos sutiles, como los dedos o la cara. Existen prototipos de sistemas de este estilo en la Universidad de Stanford, en el Instituto Tecnológico de Massachesetts y en la Sociedad Max Plank. También existen sistemas comerciales basados en video, consistentes en obtener datos de movimiento haciendo un seguimiento de ciertos patrones en una secuencia de imágenes.

3.3.7 Captura mediante fibra óptica. Los primeros sistemas de este estilo son los guantes de fibra óptica, están constituidos por un conjunto de fibras ópticas que, al doblarse, atenúan la luz transmitida, permitiendo calcular la posición de los dedos de la mano. El primer ejemplo de este sistema es el Dataglove, para capturar los movimientos del cuerpo, se fijan sobre distintas partes del cuerpo sensores flexibles de fibra óptica que miden las rotaciones de las articulaciones. Al igual que los sistemas electromecánicos, no se mide la posición del actor en el escenario.

Figura 30. Guante de fibra óptica [61].

3.3.8 Captura mediante ultrasonidos. Se utilizan emisores que generan pulsos ultrasónicos, que son capturados por uno o varios receptores situados en posiciones conocidas, permitiendo averiguar la posición del emisor en el espacio, e incluso la orientación en algunos casos.

3.3.9 Captura mediante sistemas inerciales. Utilizan pequeños sensores (acelerómetros y giroscopios) que recogen información sobre la aceleración y la velocidad angular del sensor, conociendo estas dos variables e integrando dicha información es posible determinar la posición, eje de giro y velocidad angular de cualquier sensor; los datos adquiridos por los sensores inerciales se transmiten a un computador, donde se puede observar sobre una figura animada el movimiento completo registrado.

Figura 31. Traje con sensores inerciales y unidad motora [62].

Son difíciles de transportar y tiene grandes rangos de captura. Uno de los sistemas más conocidos de este tipo son los mando inalámbricos de Wii de Nintendo, si bien para captura de movimiento se emplean otros sensores muchos más precisos y con mayor frecuencia de captura, generalmente acoplados a unos trajes especiales con varios sensores y en donde se ubica también una unidad transmisora [63].

3.3.10 Captura óptica de movimiento pos substracción de video. Esta técnica consiste en tomar una imagen de la escena sin movimiento, a la que se llamar “fondo”, y a partir de ahí, modelar la referencia de la antigua a la nueva en la escena. Se realiza, el flujo de video que proviene de la cámara, una substracción entre cada fotograma y el fotograma

de referencia, como lo indica la figura 32. Esta resta se realiza operando pixel por pixel, de una de estas imágenes, con sus respectivos pixeles de la otra imagen. Cuando los pixeles de una y otra imagen coinciden, la substracción entre valores idénticos resulta cero, lo que traducido a color es el negro, y en el caso contrario parecen otros tonos [64].

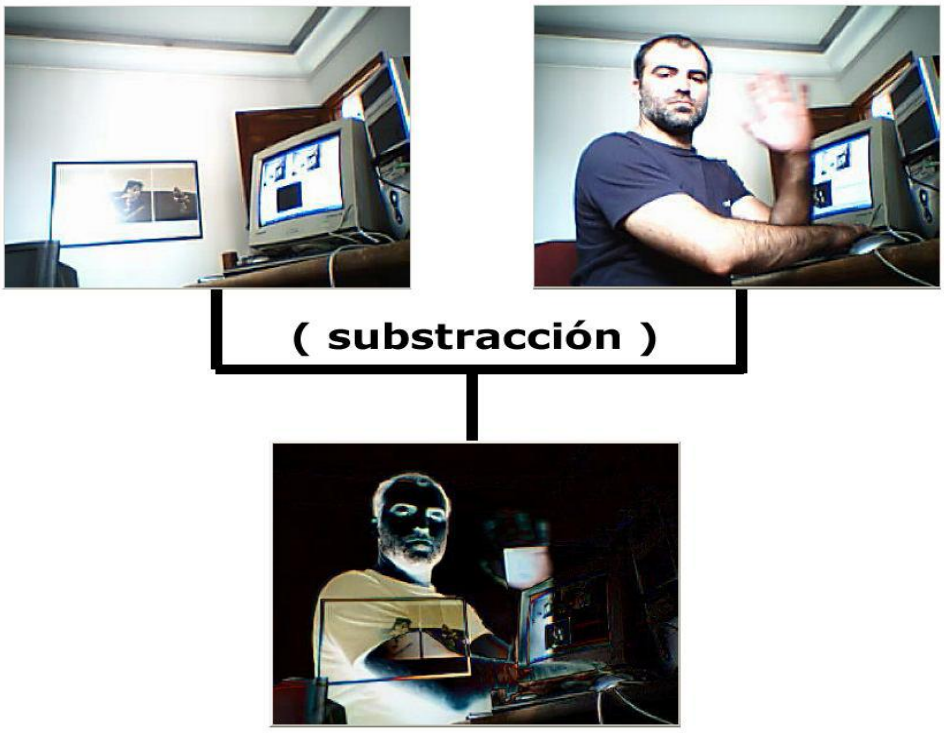


Figura 32. Captura de movimiento mediante substracción de video [65].

4. SOFTWARE ROBOT OPERATING SYSTEM ROS

Robot Operating System (ROS) es una plataforma software, lo que se conoce como framework, para el desarrollo de software específico para robótica. Fue creado por el instituto de investigación Willow Garage en 2007bajo licencia BSD y todos sus programas son de código abierto (OSS), permitiéndose así su uso gratuito tanto para la investigación como para fines comerciales.

Su filosofía es la que de todos los programas sean de uso libre, reutilizables, escalables según las aplicaciones deseadas e integrables con todo el software de robótica existente y futuro. Para ello el sistema se creó teniendo en cuenta otras plataformas de software abiertas ya existentes como OpenCV, Player/S-tage, Gazebo, Orocos/KDL y otros, existiendo paquetes para su uso en ROS.

Ros se compone de un conjunto de librerías de programación, aplicaciones, drivers y herramientas de visualización, monitorización, simulación y análisis, todas reutilizables para el desarrollo de nuevas aplicaciones para robots tanto simulados como reales.

Por otro lado, ROS proporciona servicios estándares propios de un sistema operativo pero sin serlo, ya que se instala sobre otro, en general Linux y de manera recomendada Ubuntu, por ello también recibe la denominación de Meta-Sistema Operativo. Posee su propio manejador de paquetes mediante comandos desde terminal para la gestión, compilación, y ejecución de archivos, así como la abstracción de hardware.

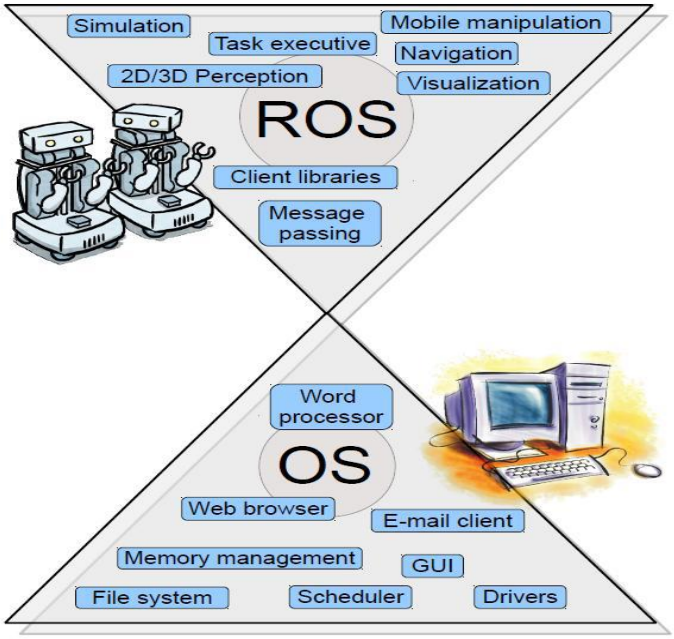


Figura 33. Relación ROS y SO [66].

Su estructura es modular, de forma que cada programa o aplicación, lo que en Ros se denomina como **nodo**, se ejecuta dentro de otro ejecutable (**Master**) que funciona de núcleo del sistema y hace las veces de plataforma por la que se comunican los diferentes nodos mediante topics o servicios [67].

Dado que Ros está continuamente evolucionando, existen varias distribuciones o versiones:

- ROS Fuerte (Abril, 2012)
- ROS Electric (Agosto, 2011)
- ROS Diamondback (Marzo, 2011)
- ROS C Turtle (Agosto, 2010)
- ROS Box Turtle (Marzo, 2010)

En este proyecto se ha utilizado la distribución Hydro que corresponde a una de las últimas versiones y se ha instalado sobre Ubuntu 12.04.

4.1 CONCEPTOS BASICOS

4.1.1 Sistemas de archivos. El sistema de archivos en ROS se divide en dos niveles:

4.1.1.1 Paquetes. Son el nivel más bajo de organización del sistema de archivos de ROS. Pueden contener cualquier cosa: ejecutables (nodos), modelos, tipos de mensajes y servicios, herramientas o librerías [68].

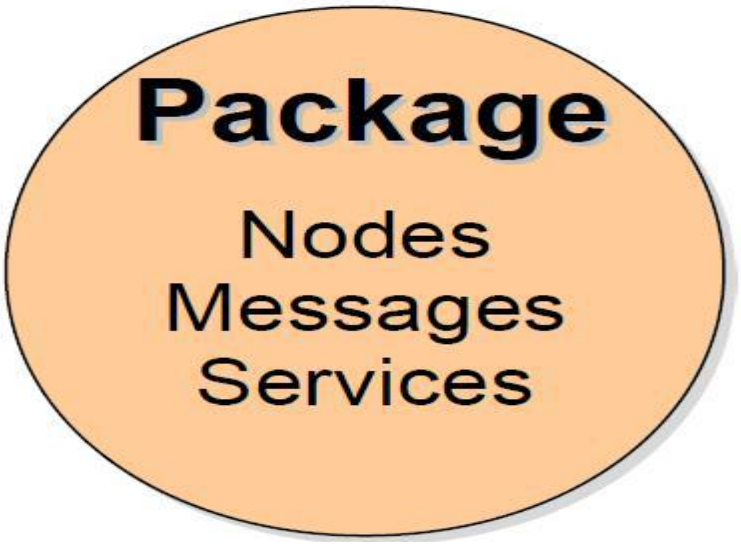


Figura 34. Paquetes [69].

4.1.1.2 Pilas (Stacks). Son agrupaciones de paquetes que forman una librería de alto nivel. Cada pila agrupa paquetes que son complementarios entre ellos.

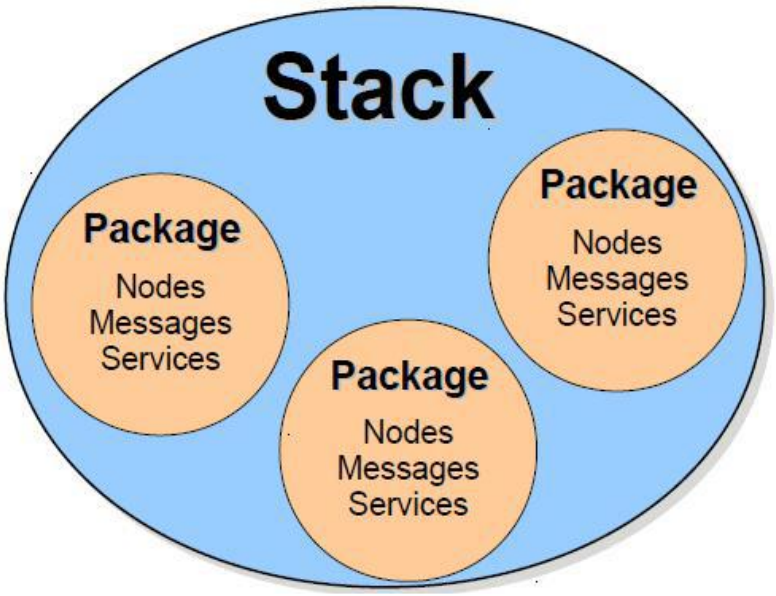


Figura 35. Pilas [70].

Ambos poseen unos archivos de información especificando lo que contienen, su funcionalidad y los paquetes de los que depende. En las pilas éste archivo se denomina stack.xml y en los paquetes manifest.xml. Esto es una forma rápida de diferenciar cuando si estamos dentro de un paquete o si estamos dentro de una pila.

A la hora de ser descargados e instalados, las pilas se agrupan en repositorios tal como se muestra en la figura 36, que son equivalentes a los repositorios de Linux. De hecho los repositorios de ROS pueden ser descargados equivalentemente a los de Linux, directamente por la línea de comandos [71].

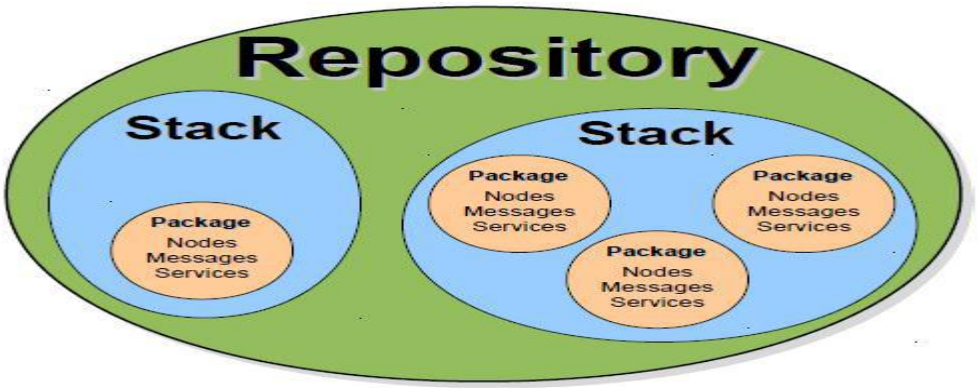


Figura 36. Repositorios [72].

Por defecto, todo el sistema ROS se instala en la dirección /opt/ros estando restringidos los derechos de modificación de los paquetes para evitar dañar el sistema.

4.1.2 Componentes de los paquetes. En un paquete de ROS encontramos las siguientes carpetas:

- Carpeta bin: Contiene los ejecutables del paquete, los que se podrán lanzar como nodos
- Carpeta build: Contiene los archivos de compilación internos del paquete
- Carpeta src: Contiene los códigos de los programas (.cpp), que al compilarlos crearan ejecutables en bin.
- Carpeta include: contiene los archivos de las cabeceras (.h), propios de los programas del paquete
- Carpeta lib: Contiene los archivos de las librerías (.lib, .so)
- Carpeta launch: Contiene los archivos de lanzamiento (.launch) de aplicaciones completas
- Carpeta msg: Contiene los tipos de mensajes (.msg) definidos en el paquete
- Carpeta srv: Contiene los tipos de mensajes de los servicios definidos en el paquete
- Archivo CMakeLists.txt: Es una lista en la que se especifica al compilador que debe compilar de nuestro paquete: nodos, mensajes, servicios, librerías.
- Archivo Manifest.xml: Contiene una descripción del paquete (función, autor, licencia, etc.) y una lista con los paquetes de los que depende.

4.1.3 Compilación. Para compilar un paquete en ROS se utiliza CMake, una plataforma de código abierto para la generación. Compilación y comprobación de paquetes de software. Su uso es bastante sencillo, basta con que el paquete de ROS contenga un archivo llamado **CMakeList.txt** compuesto por una serie de macros según lo que se quiera crear y compilar.

Los macros más comunes que se utilizan son los siguientes:

- Rosbuild_add_executable (ejecutable src/programa.cpp): crea en la carpeta bin el ejecutable de programa.cpp.
- Rosbuild_add_library (librería src/programa.cpp): crea en la carpeta lib la librería de programa.cpp
- Rosbuild_genmsg (): auto-genera todas las cabeceras de archivos necesarios de los tipos de mensaje definidos en la carpeta msg del paquete y los guardará en una nueva carpeta denominada msg_gen.
- Rosbuild_gensrv ():auto-genera todas las cabeceras y archivos necesarios de los tipos de servicios definidos en la carpeta srv del paquete y los guardará en una nueva carpeta denominada srv_gen [73].

Para compilar un paquete, basta con hacer en una terminal:

$$\$\ rosmake\ nombre\_paquete$$

Esto compilará todo lo especificado en CMakeList.txt, además de todos los paquetes de los que depende éste, es decir, aquellos especificados en manifest.xml de la forma:

$$<depend\ package=\text{“}nombre\_paquete\text{”}>$$

4.1.4 Nodos. Un nodo es un módulo o proceso individual dentro del sistema de ROS que realiza un cómputo, puede enviar o recibir información de otros nodos y usar u ofrecer servicios. Estos nodos no son más que los ejecutables de los paquetes de ROS. En la figura 34 se muestra un ejemplo de dos nodos, talker y listener en el cual el primero está publicando en un topic, denominado chatter, y el segundo está subscrito a él.

Cada nodo posee un nombre único, que lo identifica y lo distingue de todos los demás nodos en ejecución. Así por ejemplo es posible lanzar el mismo programa dos veces, lo que da lugar a dos nodos, siempre que estos tengan diferentes nombres [74].

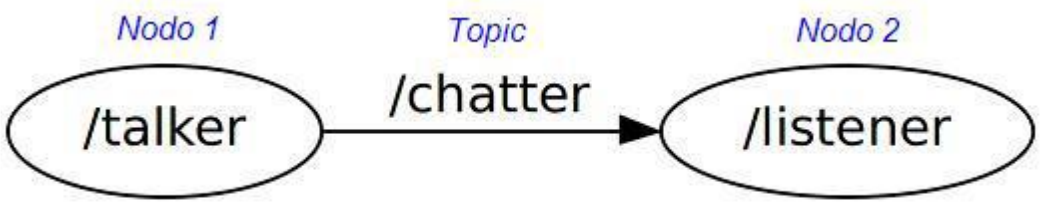


Figura 37. Nodos [75].

Los nodos se pueden escribir en C++ o en Phyton usando librería de cliente respectiva que ROS ofrece, que a su vez son paquetes de ROS:

- roscpp: librería de cliente de C++
- rospy: librería de cliente phyton

Es muy importante tener en cuenta que la ejecución y comunicación entre nodos es independiente del lenguaje en que estén escritos, funcionará correctamente siempre y cuando se cumplan los interfaces entre ellos, es decir, se envíen y reciban siempre mensajes del mismo tipo.

La programación de todos los nodos en ROS debe incluir las siguientes instrucciones:

```
#include "ros/ros.h"

int main(int argc, char **argv)
{
    ros::init(argc, argv, "nombre_nodo");
    ros::NodeHandle n;
                (...)

    return 0;
}
```

“ros.h” contiene todas las cabeceras necesarias para el uso de las instrucciones más comunes del sistema de ROS, como por ejemplo las referentes al uso de topics y servicios, pero no incluye los tipos de mensajes y otras instrucciones.

“ros::init(argc, argv, “nombre _ nodo”)” inicializa el nodo en el sistema con el nombre que le asignemos y los argumentos que haya recibido al ejecutarlo.

“ros::NodeHandle n” crea el principal punto de acceso a las comunicaciones de este nodo con el resto del sistema en ROS. Es una interfaz para crear suscripciones o publicaciones del nodo a los topics, así como para acceder al servidor de parámetros. Además, junto con init(), inicializa el nodo y lo elimina automáticamente al finalizar la ejecución del programa. “n” sería el identificado del proceso de este nodo y debe ser único

Para ejecutar ciclos a una frecuencia deseada, por ejemplo para recibir mensajes de cada cierto tiempo o controlar un robot, roscpp ofrece las siguientes instrucciones:

```
ros::Rate r(10);
while (ros::ok()))
{
    (Código a ejecutar...)

r.sleep();
}
```

“ros::Rate r(10)” permite especificar la frecuencia a la que se va a ejecutar el bucle while(). En este caso, se habría elegido una frecuencia de 10 Hz.

“ros::ok()” es una función que devuelve un valor falso cuando se dé, se reciba un [ctrl + C], haya sido expulsado el nodo porque el nombre ya existe ó “ros::shutdown()” haya sido ejecutada (por ejemplo por NodeHandle).

“r.sleep()” es la función que nos asegura que se cumpla el ciclo en 10 Hz esperando hasta el fin de éste.

4.1.5 Topics. Los topics son buses por los cuales los nodos envían o reciben mensajes. Simplemente un nodo debe comunicar al Master que desea publicar en él para enviar información o suscribirse en él para recibir información. Pueden existir varios nodos publicando y varios nodos subscritos a la vez en un mismo topic. Son anónimos, en el sentido de que los nodos publican o se suscriben a ellos pero nunca saben que otros nodos están publicando o subscritos en el topic, lo cual desacopla la producción del consumo de la información.

Todos los topics son unidireccionales, por lo cual un nodo que envía información por un topic nunca recibirá una respuesta directamente por el mismo topic ni sabrá si sus mensajes están siendo recibidos. Si se desea enviar una información y recibir una respuesta debe usarse el otro tipo de comunicación entre nodos, los servicios.

Por un topic se envían objetos de un tipo de mensaje definido en el mismo paquete o en otro paquete del que dependa el paquete en cuestión. La definición de estos mensajes-objetos se incluye en el programa mediante las cabeceras generadas en los paquetes a partir de los archivos .msg, de los que se hablará más adelante [76].

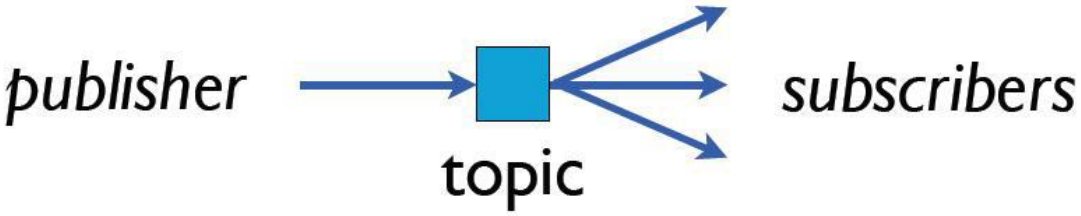


Figura 38. Topics [77].

Un topic viene identificado por su nombre, que debe ser único y normalmente es de la forma: "/nodo _ que _ lo _publica/nombre_topic", aunque esto es decisión del programador. A la hora de su declaración, además de éste nombre, es necesario definir el tipo de mensaje que se va a enviar por ellos y por lo tanto el tipo de mensaje que se va a recibir por ellos. El Master no asegura esta concordancia, por lo cual ha de tenerse especial cuidado en que al subscribirse a un topic se haga con el tipo de mensaje que se está enviando por este.

4.1.6. Servicios. Los servicios en ROS son la forma en que se envía un mensaje (request) desde un nodo a otro y éste le responde con otro mensaje (response) como lo indica la figura 39, lo que se conoce como framework, para el desarrollo de software específico para robótica [78].

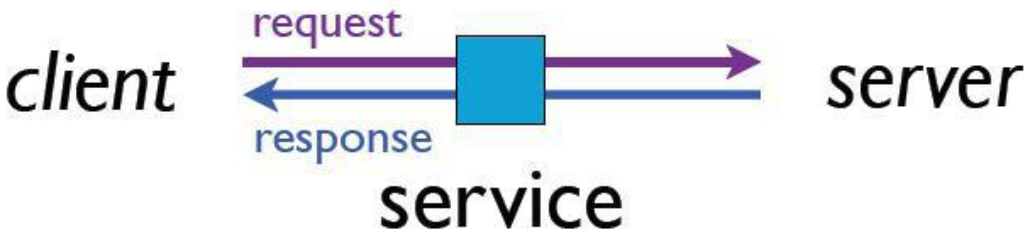


Figura 39. Servicios [79].

Así un nodo puede ofrecer un servicio (server) y cualquier otro nodo utilizarlo (client) llamándolo junto con un mensaje y esperando una respuesta de éste, es decir, con otro mensaje.

El servicio ofrecido por un nodo es totalmente independiente de los clientes. Es posible, por ejemplo, que un nodo ofrezca un servicio y el usuario llame a este servicio desde una terminal.

Los servicios vienen definidos mediante los archivos .srv, que definen el tipo de servicios que se van a ofrecer, esto es el tipo de mensaje que se va a recibir como request, el tipo de mensaje que se va a devolver como response. Al igual que con los mensajes para los topics, ahora se deben incluir las cabeceras auto-generadas por los archivos .srv, que contendrán las definiciones de las clases del tipo de servicio.

4.1.7 Mensajes. Los nodos se comunican entre sí publicando y recibiendo mensajes vía topics. Los mensajes son estructuras de datos equivalentes a una estructura de C++ que se definen en la carpeta msg de los paquetes mediante archivos de texto con la extensión .msg, en forma de lista tipo-nombre:

Mensaje:	
	type1 name1 type2 name2 type3 name3

La estructura puede estar formada por primitivas estándares equivalentes a las de C++, definidas en el paquete std_msg y que se describen en la tabla 1.

msg	C++
bool	uint8_t
int8	int8_t
uint8	uint8_t
int16	int16_t
uint16	uint16_t
int32	int32_t
uint32	uint32_t
int64	int64_t
uint64	uint64_t
float32	float
float64	double
string	std::string
time	ros::Time
duration	ros::Duration

Tabla 1. Equivalencia de tipos de datos.

También pueden estar formados por otros tipos de mensajes (.msg) estándares o ya definidos en std_msg o en cualquier otro paquete de ROS, siempre y cuando exista la dependencia oportuna entre paquetes.

4.1.8 Master. El Master (o roscore) es un nodo que funciona de núcleo del sistema, proporcionando una plataforma mediante el registro de nombres, servicios y parámetros sobre la cual se hace posible la comunicación y el envío de datos entre los diferentes nodos individuales del sistema. Sin él los nodos no se "encontrarían" entre ellos [80].

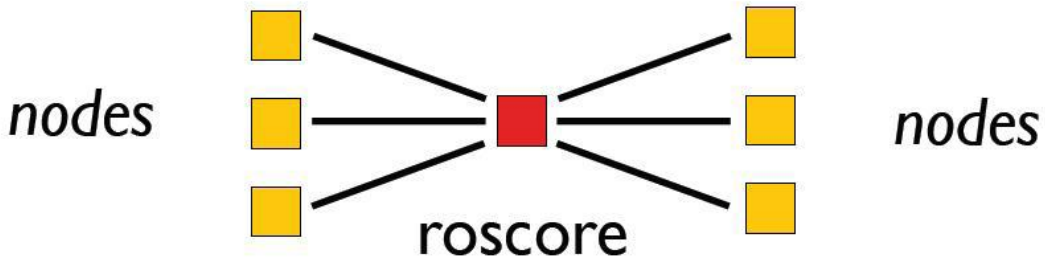


Figura 40. Nodos [81].

Realmente Roscore es una herramienta que proporciona tres cosas:

- El Master, entendido como plataforma de comunicación para los nodos.
- El servidor de parámetros
- Registro de salida al que todos los nodos envían información denominado /rosout

La plataforma de comunicación se describe en la figura 41:

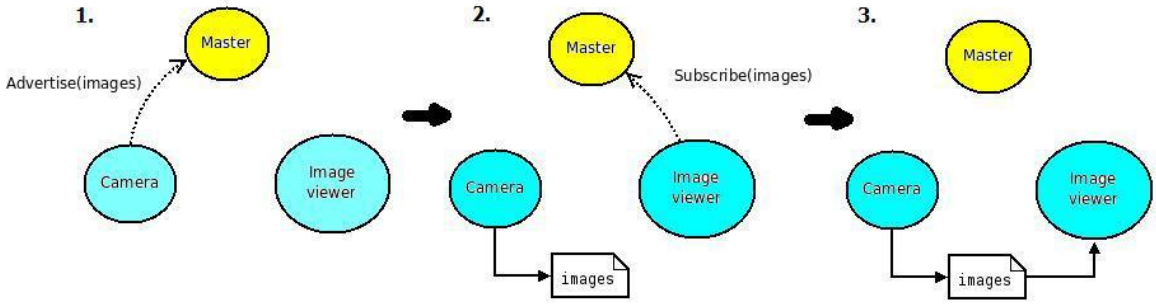


Figura 41. Funcionamiento del Master [82].

El servidor de parámetros, éste es utilizado por los nodos para almacenar o leer parámetros en tiempo de ejecución, haciendo además posible que el usuario pueda ver o modificar estos parámetros por terminal en tiempo de ejecución.

La ejecución del Master es obligatoria antes de lanzar los nodos que se comuniquen entre ellos o necesiten leer parámetros, es decir, siempre. Para ello basta lanzar por terminal:

```
$ roscore
```

4.1.9 Lanzadores. Si se desea lanzar varios nodos a la vez, cargar una serie de parámetros o ejecutar varias herramientas, existe en ROS una herramienta denominada **roslaunch,** que permite fácilmente lanzar múltiples nodos con sus argumentos, establecer una serie de parámetros en el servidor, se puede lanzar desde un terminal así:

```
$ roslaunch nombre_paquete nombre_archivo.launch
```

Los lanzadores .launch son archivos de configuración escritos en XML en los que se especifican los nodos a lanzar, los argumentos de entrada a éstos y los parámetros necesarios para crear una aplicación completa. Como todo archivo XML, se compone de una serie de etiquetas (tags). A continuación comentan las etiquetas más comunes:

- <launch>: Es el primer elemento de cualquier archivo .launch. Es simplemente un identificador del tipo de archivo del que trata. Siempre se encontrara al principio y al final en formato de cierre. </launch>.

- <node>: Se utiliza para lanzar un nodo. En principio, el nodo es una copia del nodo “tipo” que se quiere lanzar por lo que hay que darle un nombre diferente y especificar el paquete y el nombre del nodo “tipo”:

```
<node pkg="tortuga_pkg" name="tortuga" type="tortuga_node"
args="-v 10" respawn="true"/>
```

Los dos últimos son argumentos opcionales, args y respawn.

- <include>: permite incluir otro archivo launch (.launch o .xml) a partir de su ruta:

```
<include file="$ (find paquete que lo contiene)/ ruta hacia él/ achi-
vo.xml">
```

- <param>: permite definir un parámetro en el Servidor de parámetros.

Este parámetro puede ser un valor numérico constante:

```
<param name="P" value="3.4">
```

O una cadena de caracteres en la que se almacena el contenido de un archivo:

```
<param name="robot_description" command="$(find quadrotor/ro-
bots/quardrotor.urdf)">
```

- <arg>: Permite recibir argumentos al ejecutar el launcher y definirle unos valores por defecto:

```
<arg name="prueba" default="true" />
```

4.2 HERRAMIENTAS DE LÍNEAS DE COMANDOS. ROS tiene sus propias herramientas de líneas de comandos para la búsqueda de paquetes y archivos, la compilación y depuración de código. A continuación se presentan lo más usuales:

- $ roscore: ejecuta el Master, el servidor de parámetros y el nodo de registros.
- $ rosstack: obtiene información sobre la pila. Uso:

```
$ rosstack [comando] [stack]
```

- $ rospack: Obtiene información sobre el paquete en ROS. Uso:

$ rospack [comando] [paquete]

- $ roscd: acceso a un paquete.Uso:

$ roscd [paquete]

- $ rosls: Se observa el contenido de paquetes. Uso:

$ rosls [paquete]

- $ roscreate-pkg: crea un paquete en el lugar donde se encuentre el usuario, con posibilidad de incluir las dependencias entre otros paquetes directamente. Uso:

$ roscreate-pkg [nombre paquete] [dependencia1] ...

- $ rosmake: compila el paquete deseado y todos los que dependan. Uso:

$ rosmake [paquete]

- $ rosdep: descarga e instala los paquetes de los que dependa el paquete en cuestión. Uso:

$ rosdep install [paquete]

- $ rosrun: ejecuta un nodo individual. Uso:

$ rosrun [paquete] [ejecutable]

- $ rosnode: obtiene la lista de nodos en ejecución, Uso:

$ rosnode [comando] [nodo]

- $ rostopic: Obtiene la lista de los topics en ejecución. Uso:

$ rostopic [comando] [topic] [argumentos]

- $ rosservice: Obtiene la lista de los servicios de los nodos en ejecución. Uso:

$$\$\ rosservice\ [comando]\ [servicio]\ [argumentos]$$

- $ rosmsg: Obtiene la información de un tipo de mensaje. Uso:

$$\$\ rosmsg\ [comando]\ [tipo\_mensaje]$$

4.3 TRABAJANDO EN ROS. Para este proyecto se trabajara sobre la distribución Hydro de ROS, la cual se desarrollará sobre Build por efectos de compilación de archivos provenientes de C++. A continuación se describirá todo lo relacionado con la creación de paquetes, mensajes, nodos, etc. Todo lo que corresponda al buen funcionamiento de ROS, C++ y Kinect.

4.3.1 Creación de un Workspace. Un workspace en ROS se denominará un espacio de trabajo, se denominará una carpeta principal la cual contiene subcarpetas que comprenden archivos de compilación, mensajes y servicios para ser utilizados en ROS, lo primero es declarar en la terminal el código para su creación, como se muestra en la figura 42.

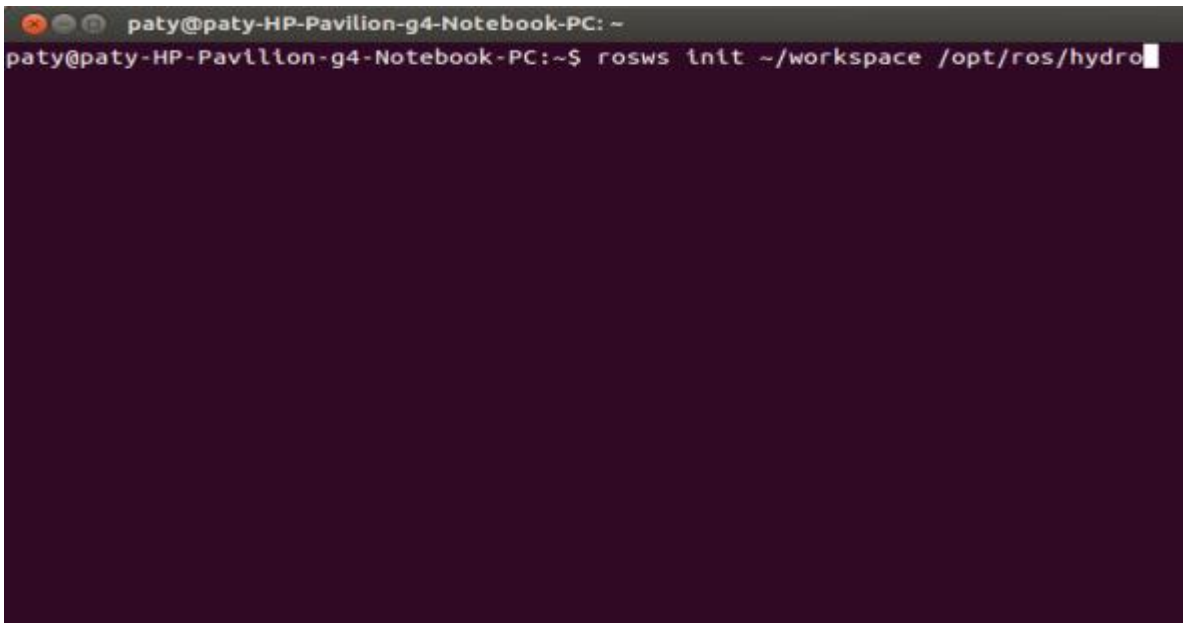


Figura 42. Creación de un workspace.

Seguidamente, se debe verificar la creación del espacio de trabajo en la carpeta personal o home del PC, como se relaciona en la figura 43.

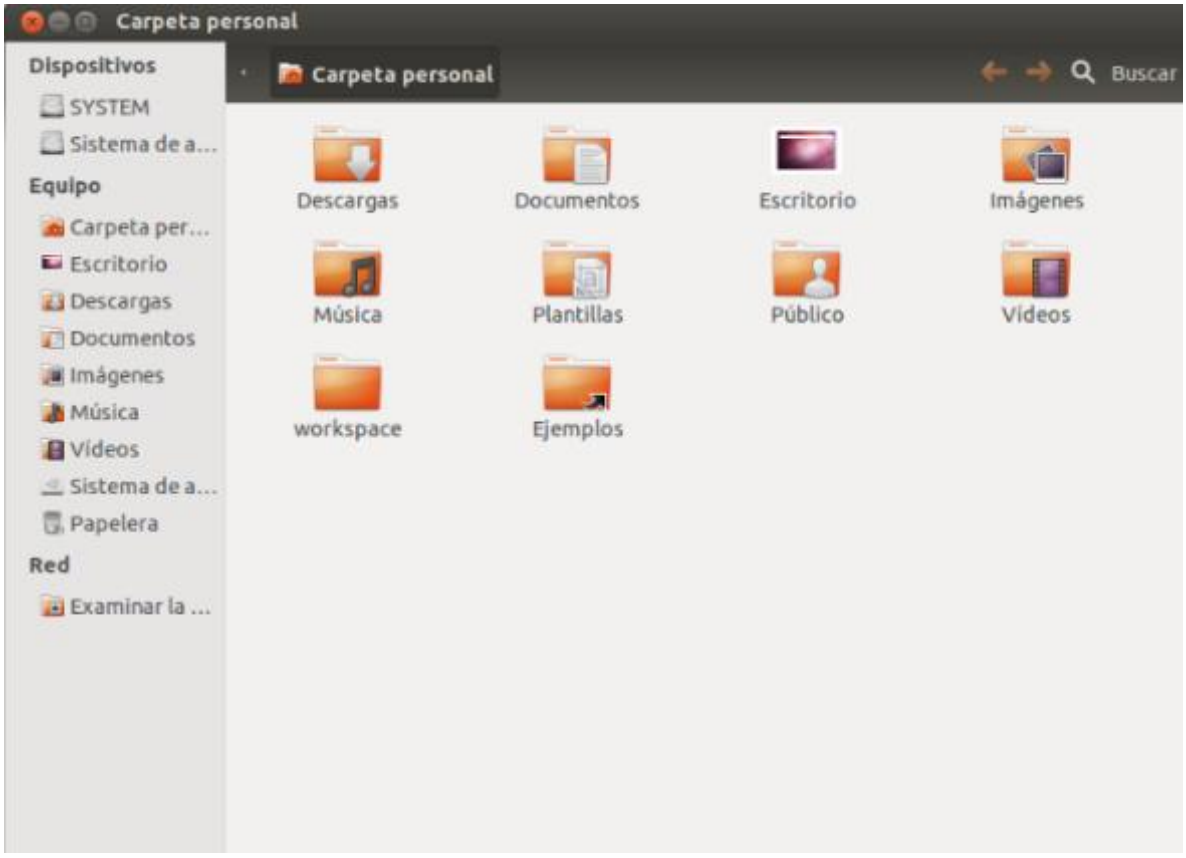


Figura 43. Verificación de workspace.

Una vez verificada su creación, se deberá abrir dicha carpeta para tener la certeza que los archivos de compilación requeridos se hayan creado, con el fin de trabajar más adelante con C++ y establecer una comunicación entre los dos lenguajes para interactuar con el Kinect.

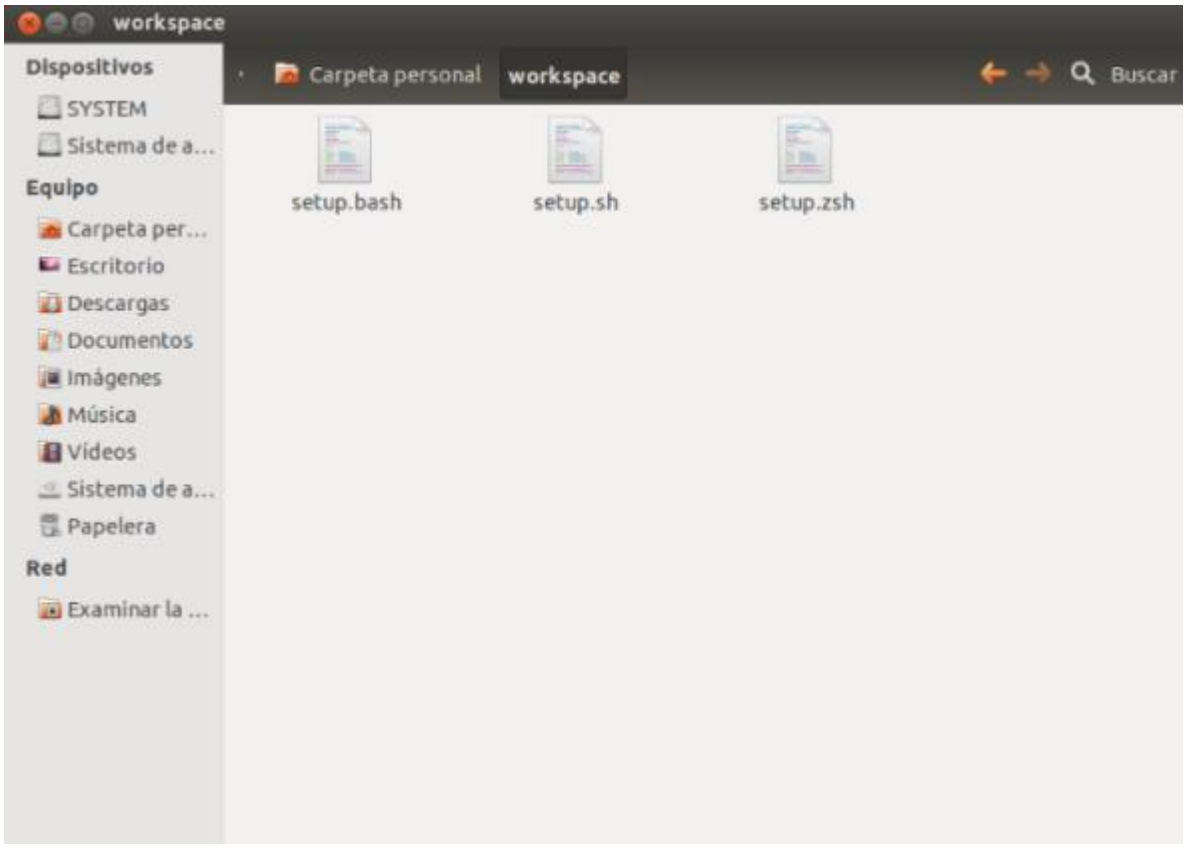


Figura 44. Verificación de Setup.

4.3.2. Creación de un directorio para un nuevo paquete. El directorio del paquete nuevo estará comprendido en el espacio de trabajo principal llamado workspace. Para su creación se deberá abrir la terminal y ejecutar el siguiente código, con el cual se creará un directorio para un nuevo paquete llamado sandbox, como se muestra en la figura 45.

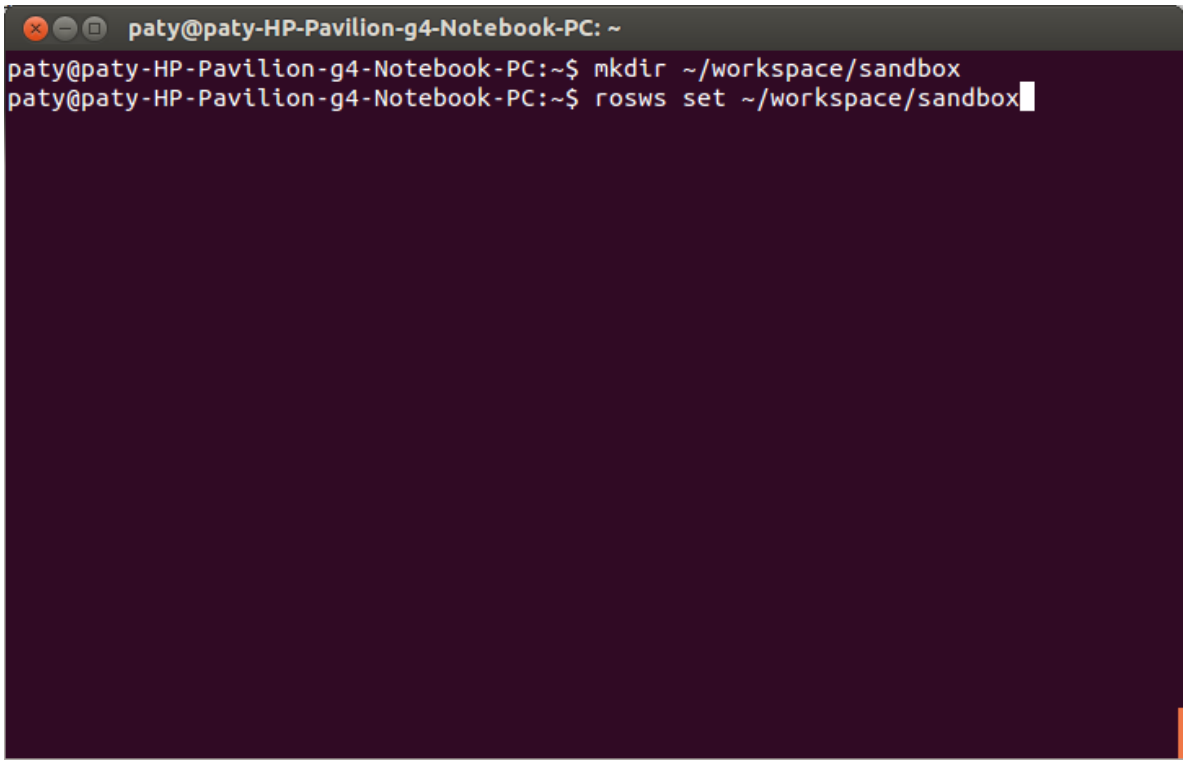


Figura 45. Creación de una dirección.

Para verificar la creación de la dirección del nuevo paquete, se deberá acceder a la carpeta personal o home y visualizarlo, como se indica en la figura 46.

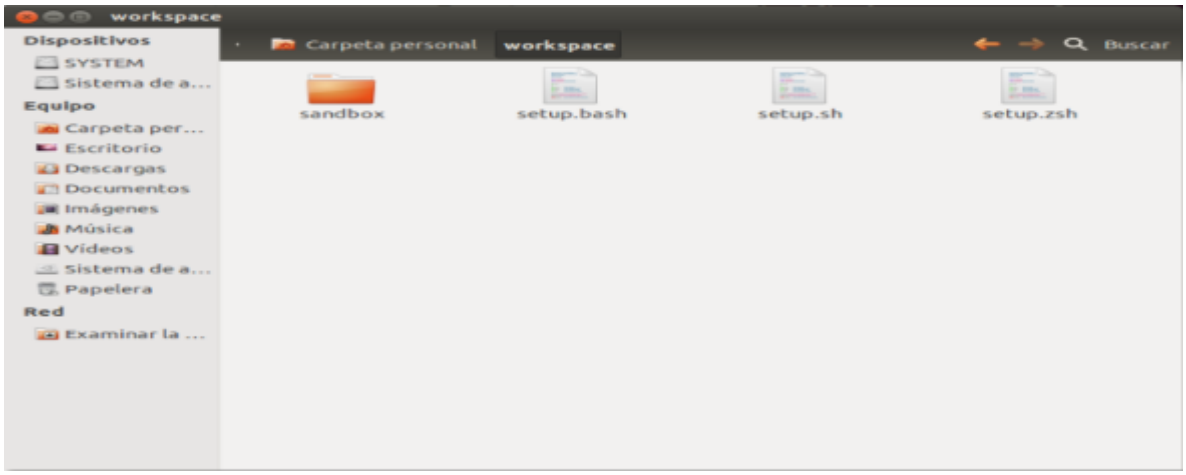


Figura 46. Verificación de la creación de dirección de un paquete en ROS.

4.3.3 Creación de un nuevo paquete en ROS. Una vez creada la dirección se procederá a establecer el nuevo paquete que contendrá información importante para la ejecución del proyecto tanto en ROS como en C++, se debe acceder a la dirección de sandbox, creada anteriormente y ejecutar en la terminal la siguiente instrucción donde se relaciona la creación del paquete y se llaman las dependencias como mensajes y los archivos para lenguajes Phyton y C++, como se establece en la siguiente figura 47.

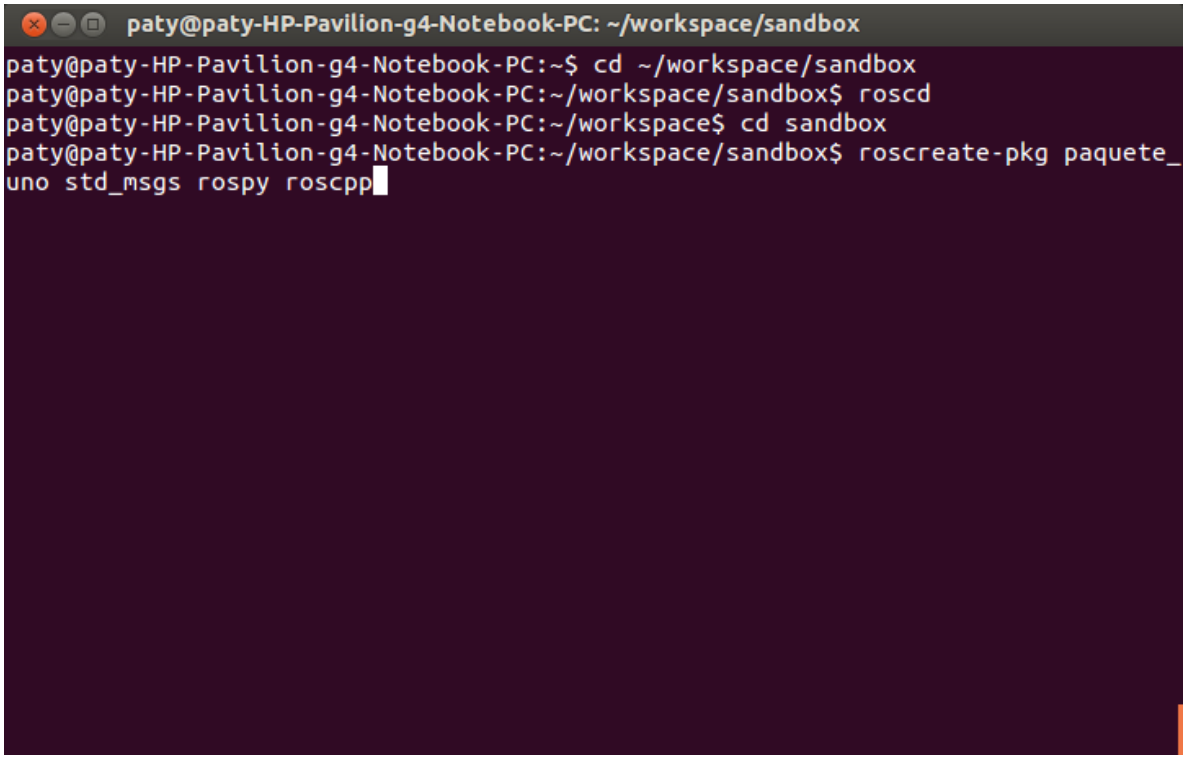


Figura 47. Creación de un paquete en ROS.

Con la creación del nuevo paquete se deberá observar en la terminal la apertura de archivos como: src, Makefile, manifest.xml, CMakeList.txt, mainpage.doc, necesarios para desarrollar aplicativos de servicios, mensajes y nodos, a continuación se muestra según la figura 48.

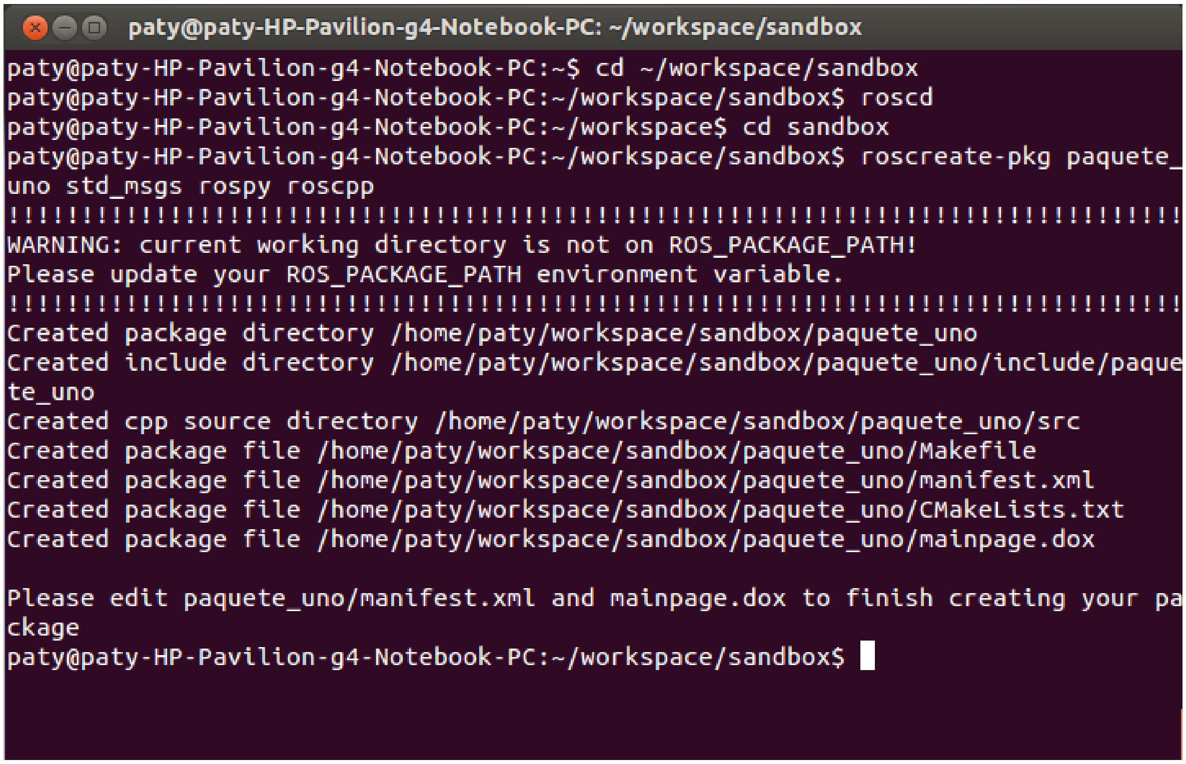


Figura 48. Creación de archivos que componen un paquete en ROS.

Dentro de la carpeta sandbox creada anteriormente se deberá verificar la creación del paquete, como se muestra en la figura 49.

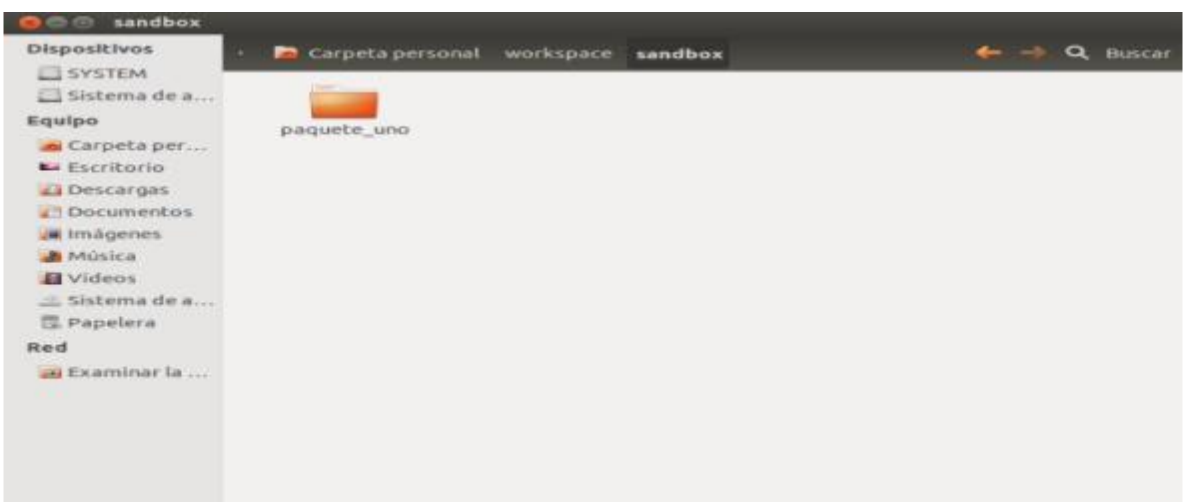


Figura 49. Creación de paquete.

En este paquete deberá aparecer los archivos creados que se visualizaron en la terminal, los cuales marcan la diferencia de la creación de un espacio de trabajo y una dirección, como se relacionan en la figura 50.

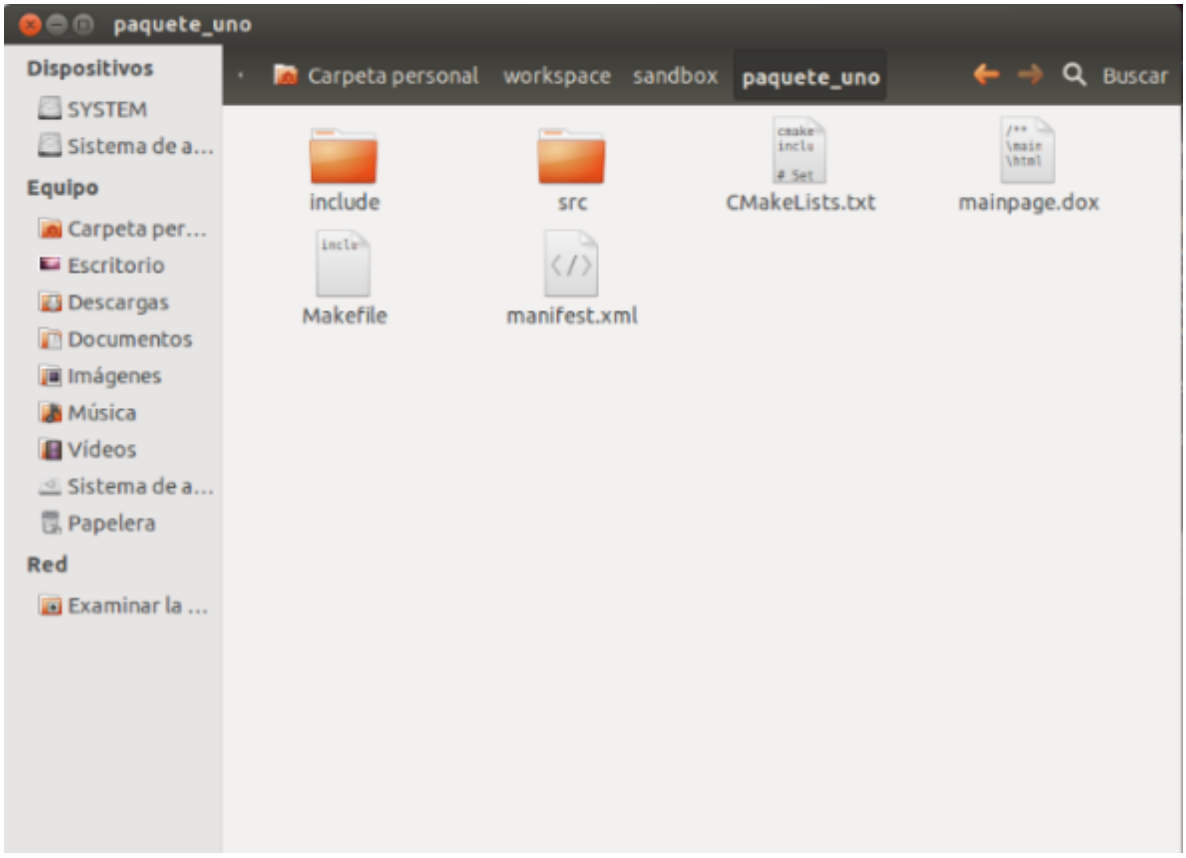


Figura 50. Archivos que componen el paquete en ROS.

Para verificar el perfil del paquete creado, se deberá realizar en la terminal la ejecución de la siguiente instrucción, y se relacionará los directorios utilizados, como se muestra en la figura 51.

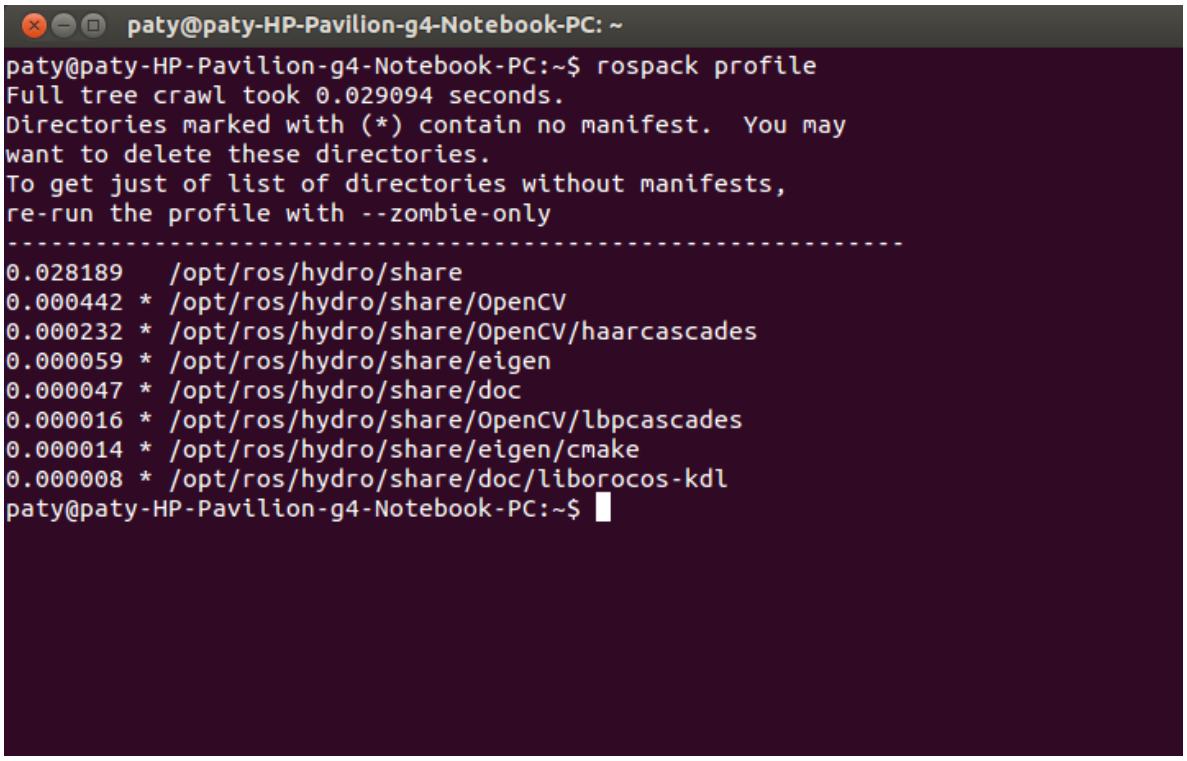


Figura 51. Verificación de perfiles.

También se puede verificar las dependencias que están declaradas en el paquete, con la instrucción $cat manifest.xml, se podrá observar como lo indica la figura 52.

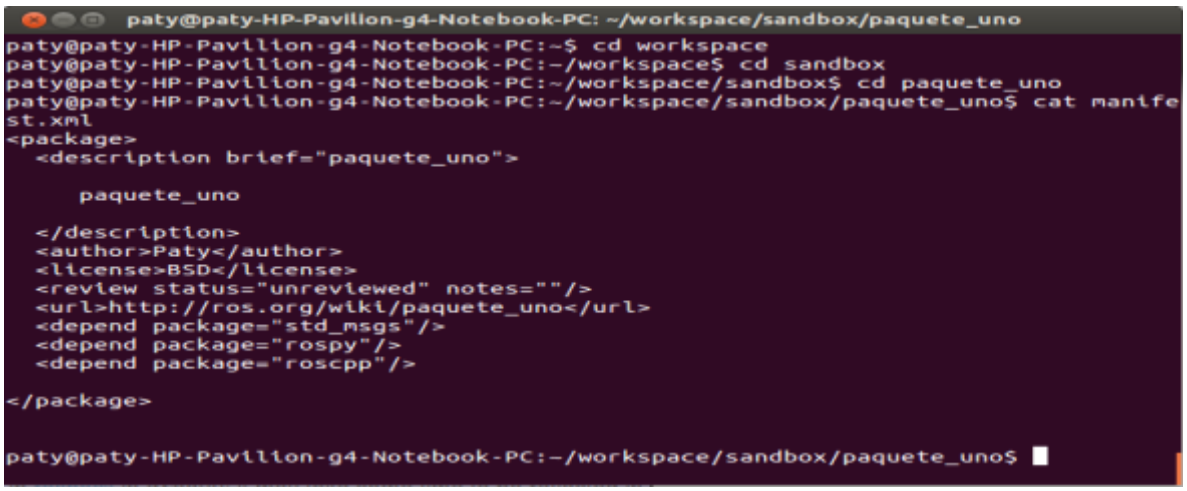


Figura 52. Verificación de dependencias.

4.3.4 Inicializando ROS. Para correr ROS se deberá abrir una terminal y ejecutar el código roscore, como se muestra en la figura 53.

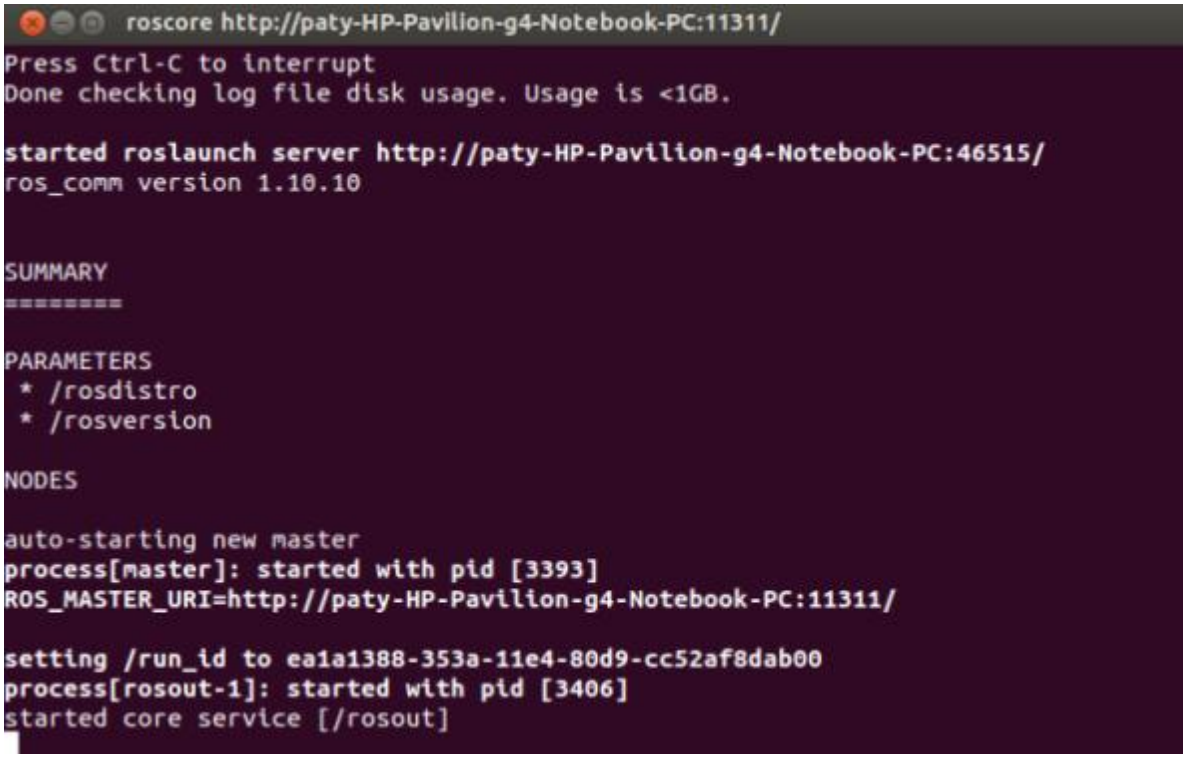


Figura 53. Roscore.

En otra terminal se podrá verificar la lista de nodos activos, se podrá visualizar con la siguiente instrucción.

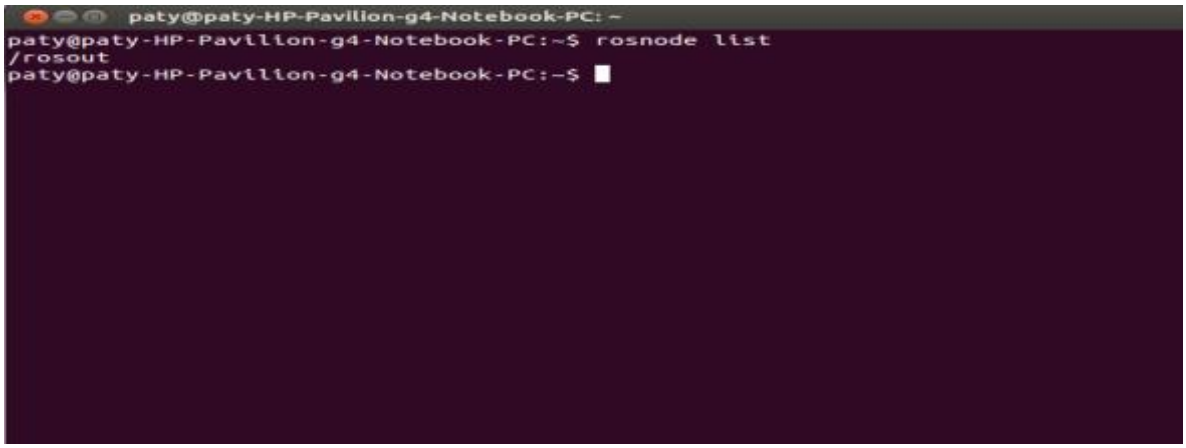


Figura 54. Código para listar nodos en terminal.

Se visualizará por terminal los nodos activos según la aplicación que se esté ejecutando en ROS, como se muestra en la figura 55.

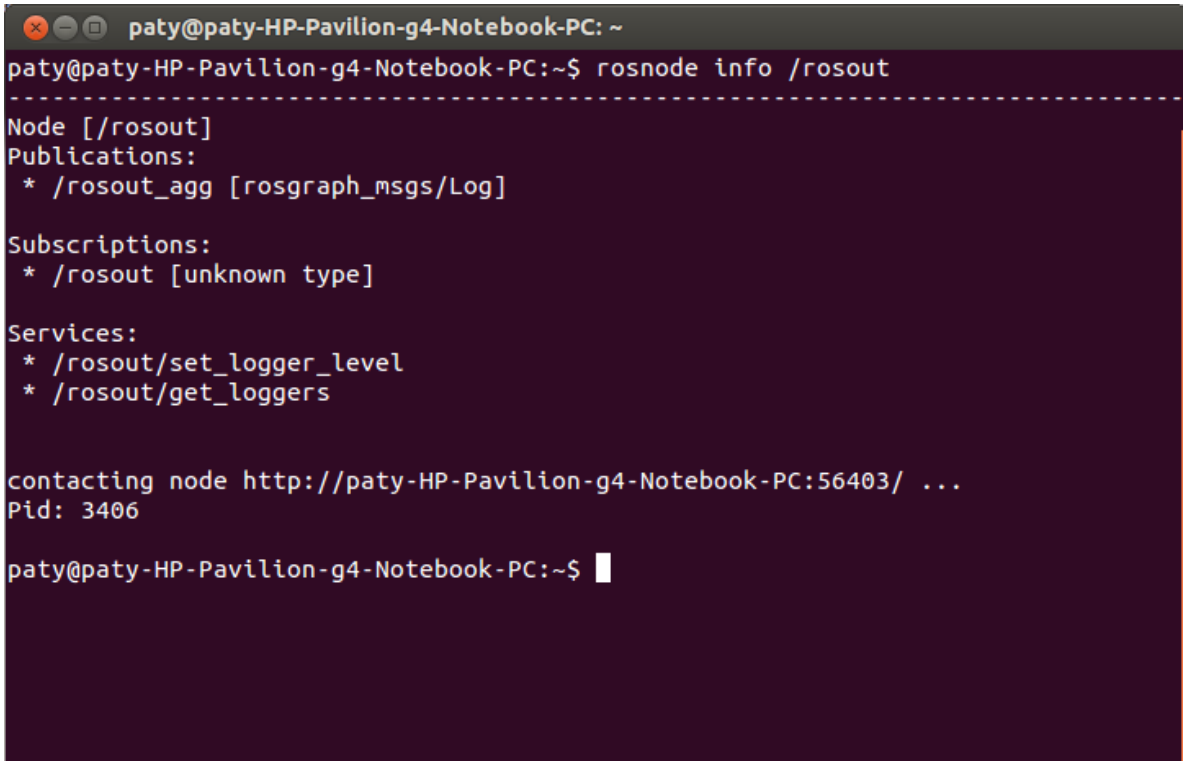


Figura 55. Vista en pantalla de nodos activos.

4.3.5 Creación de MSG Y SRV (mensajes y servicios). Para la creación de mensajes y servicios en ROS, se debe acceder al paquete creado, para este caso paquete_uno y crear con el siguiente código un mensaje como lo indica la figura 56.

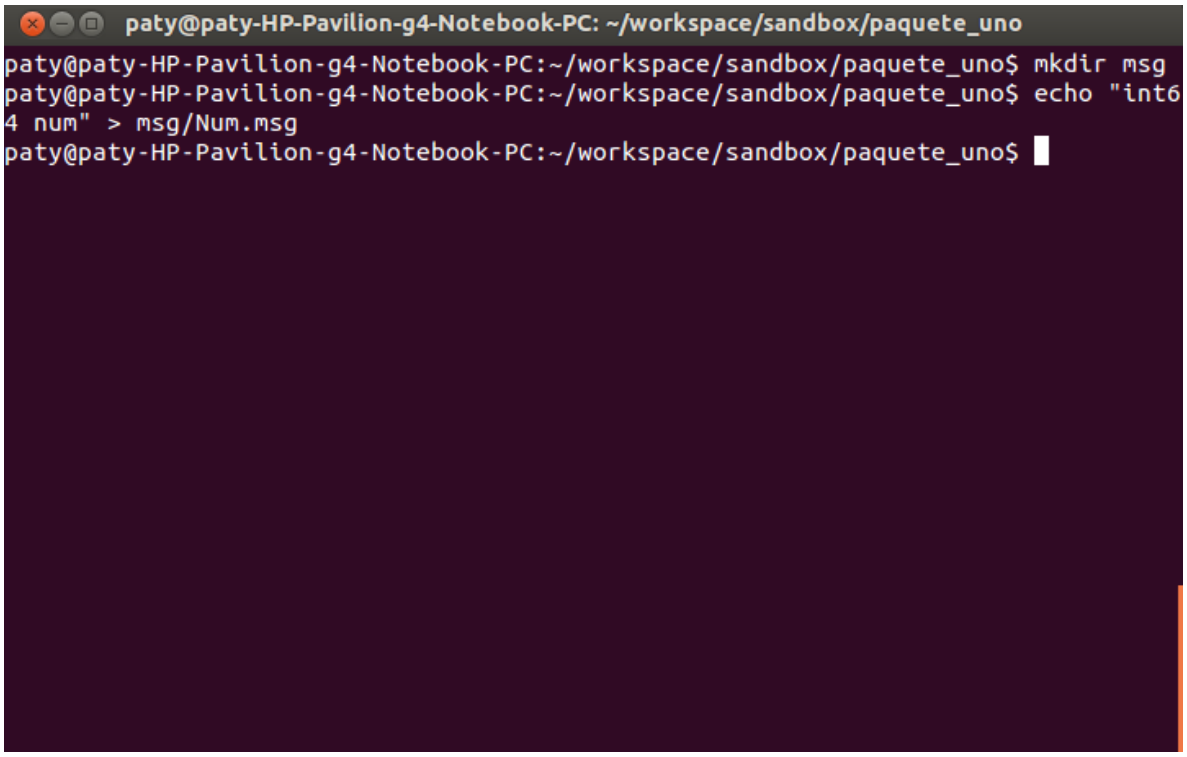


Figura 56. Creación de un mensaje en ROS.

Seguidamente se debe crear una carpeta llamada srv dentro de paquete_uno la cual contendrá el servicio que se quiere realizar, como se indica en la figura 57.

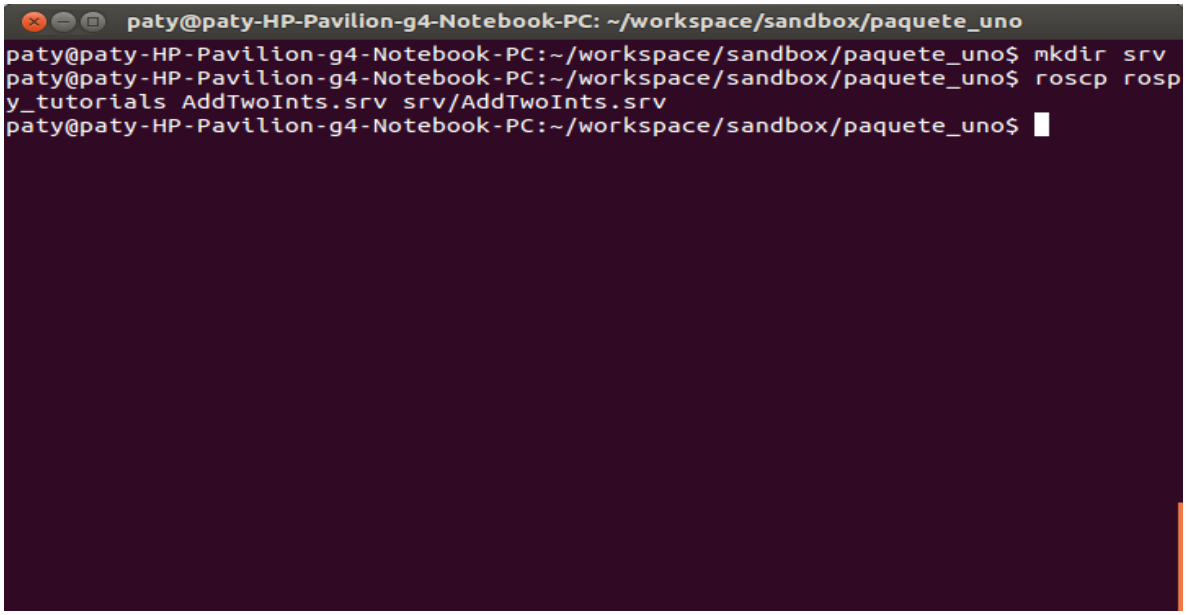


Figura 57. Creación de un servicio en ROS.

Para verificar la creación de estas dos carpetas msg y srv en paquete_uno, se debe acceder al home del PC, así como se indica en la figura 58.

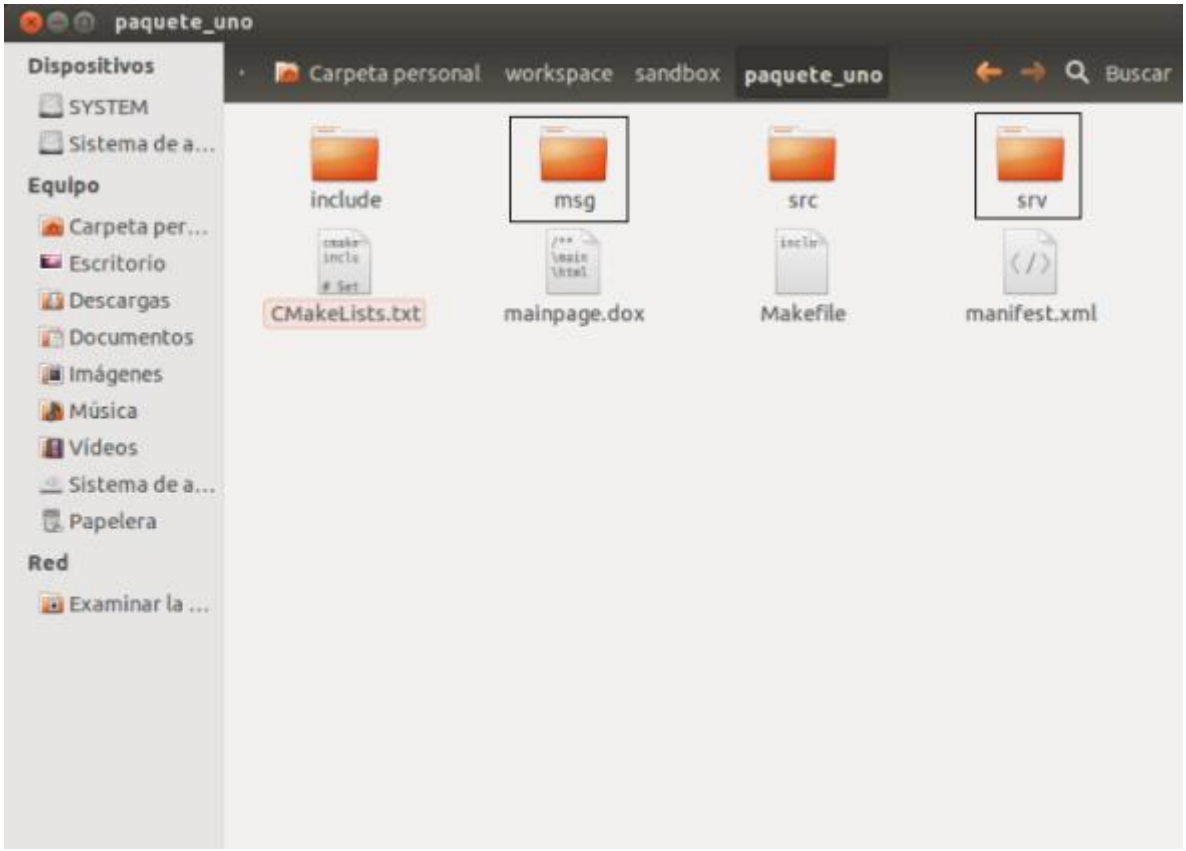


Figura 58. Carpetas msg y srv.

Seguidamente se procederá a verificar lo que está cargado en la carpeta msg, como se indica en la figura 59.

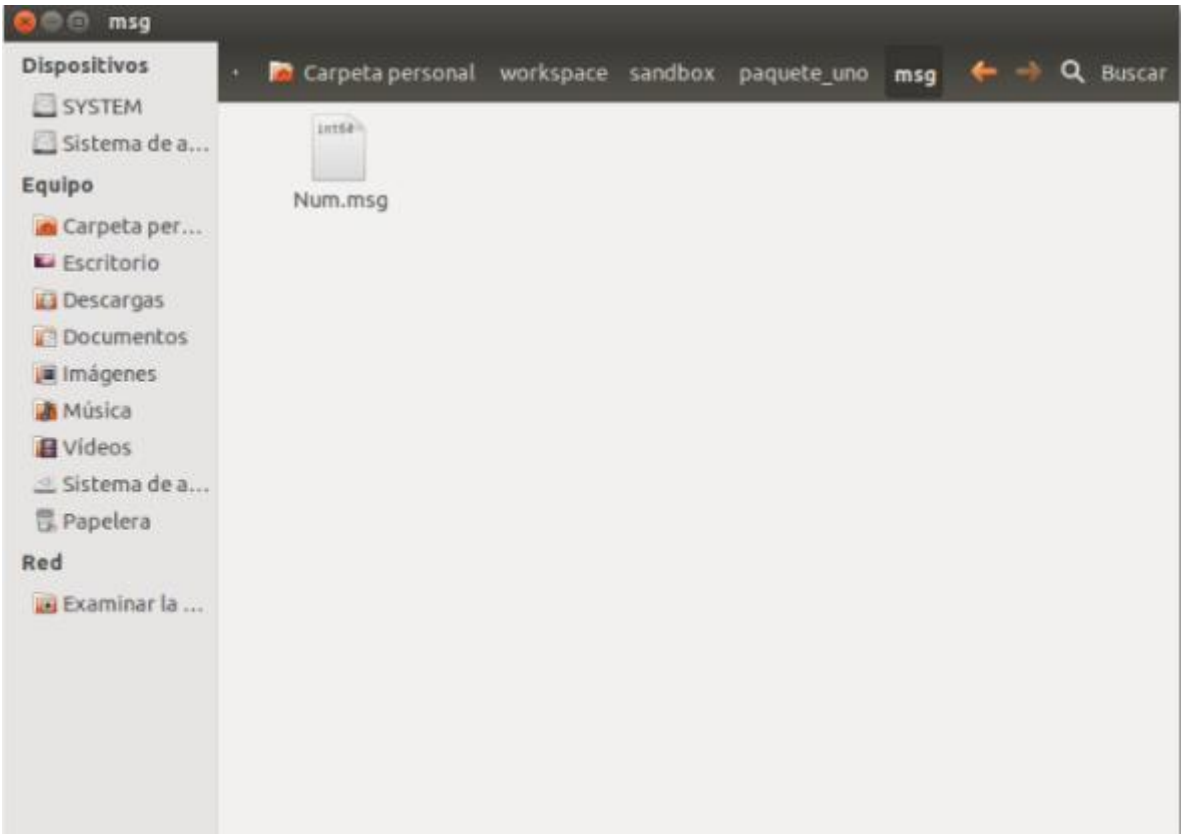


Figura 59. Mensaje cargado en la carpeta msg.

Se puede verificar lo que está cargado en el archivo usando el gedit de Ubuntu, como se muestra en la figura 60.

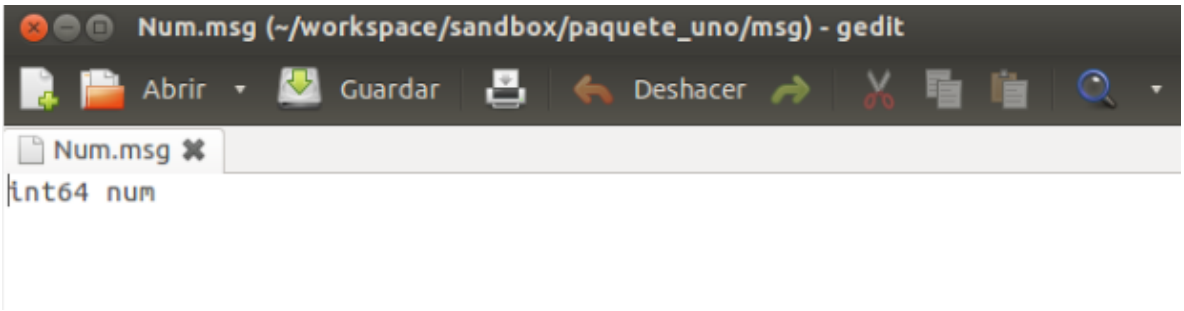


Figura 60. Gedit del mensaje cargado en msg.

Se procederá a verificar el servicio cargado en la carpeta srv como se indica en las figuras 61 y 62.

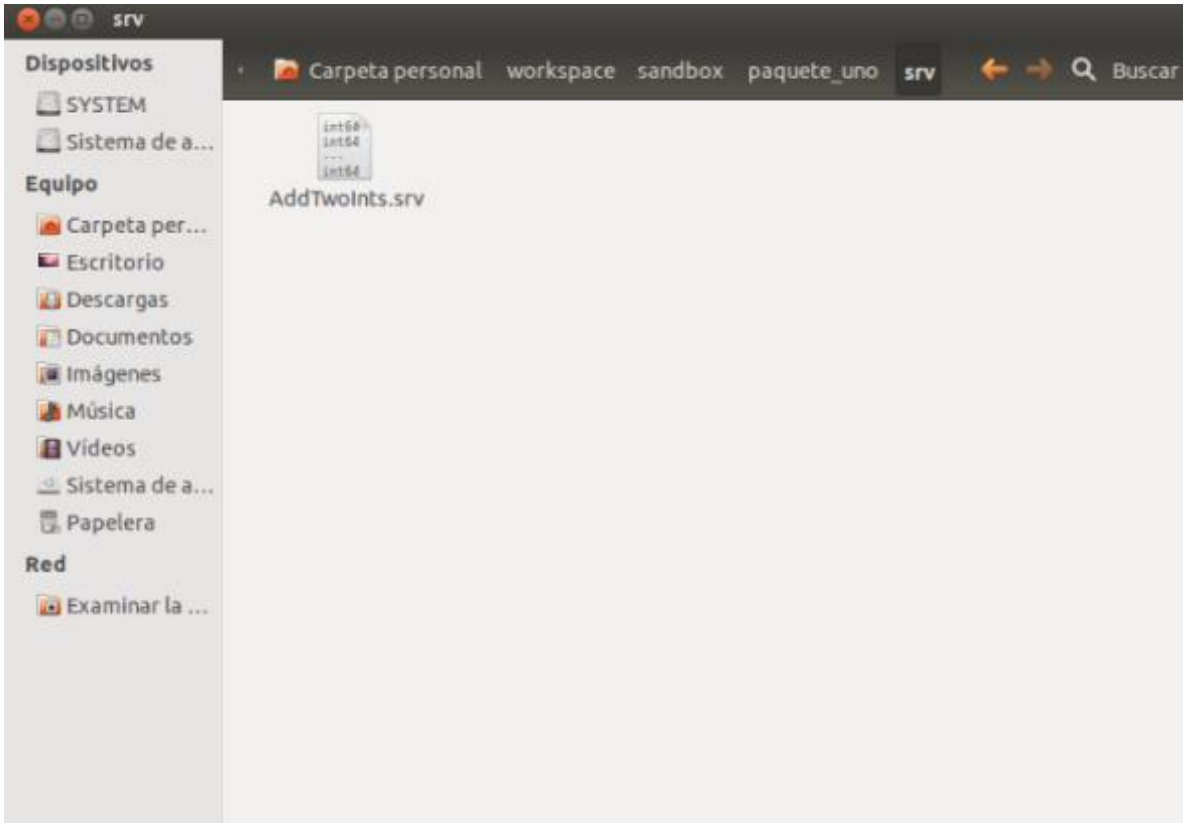


Figura 61. Servicio cargado en la carpeta srv.

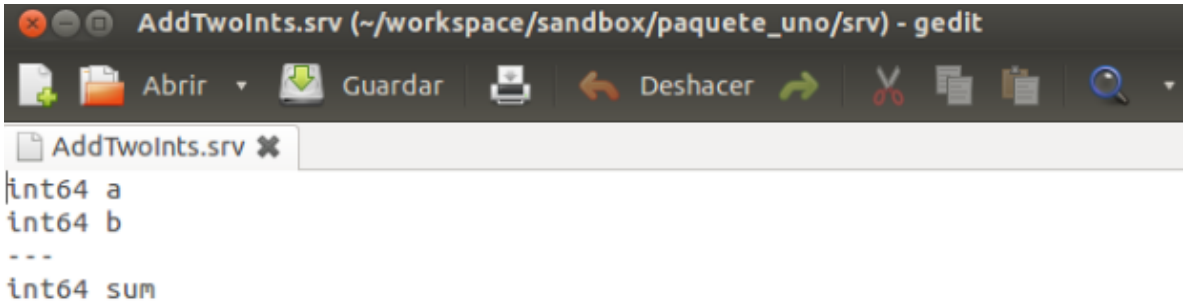


Figura 62. Gedit del servicio cargado.

Para visualizar en pantalla el mensaje cargado anteriormente, se deberá abrir la terminal de Ubuntu con la siguiente instrucción como lo indica la figura 63.

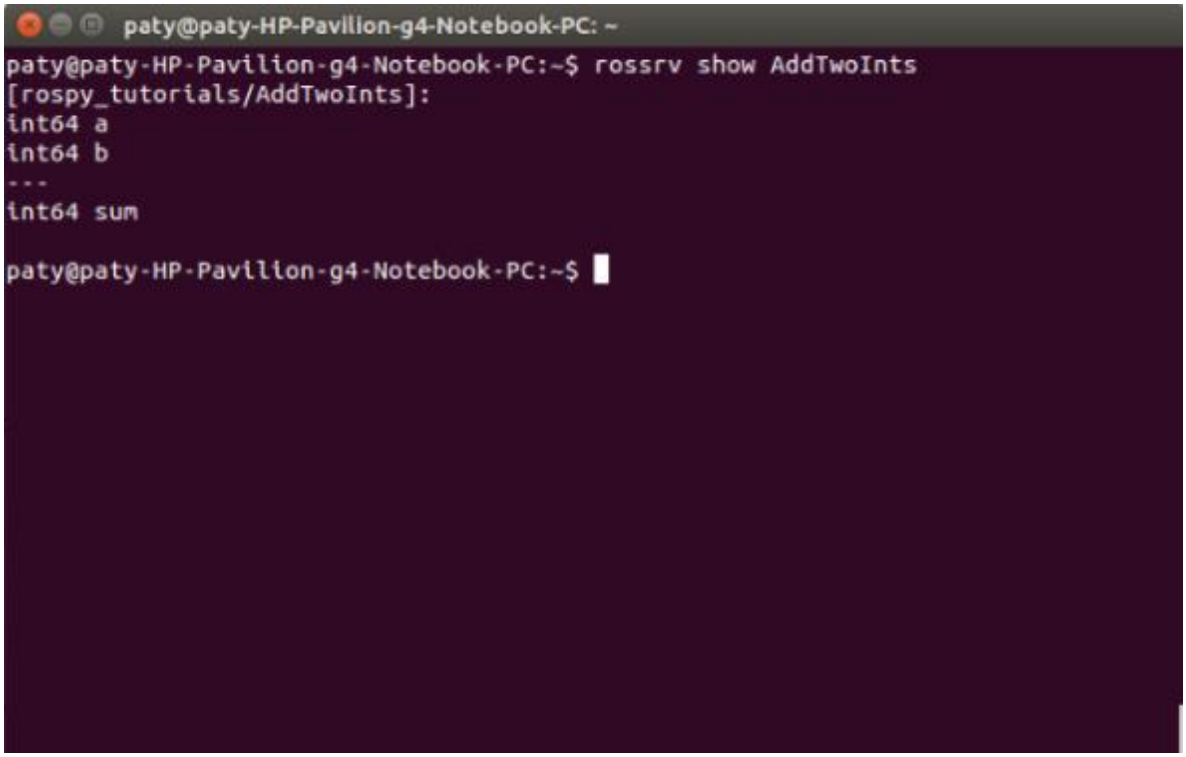


Figura 63. Mensaje en pantalla.

4.3.6 Manejo del Kinect en ROS. Debido a la finalidad de este proyecto se utilizaron varios paquetes que contiene ROS como el OpenNI_Launch, OpenNI_Tracker y Skeleton Marker, relacionados con la imagen RGB y esqueletización de usuario.

4.3.6.1 OpenNI_Launch. Este paquete contiene los archivos de lanzamiento para el uso de OpenNI utilizando Kinect bajo el software ROS. Crea un nodo el cual transforma lo datos desde el controlador en nube de puntos e imágenes necesarios para el procesamiento y visualización.

A partir de la versión de ROS Hydro, toda la funcionalidad de OpenNI_Launch ha sido movida a rgbd_launch, con el fin de permitir que otros drivers puedan utilizar el mismo código.

- Lanzamiento

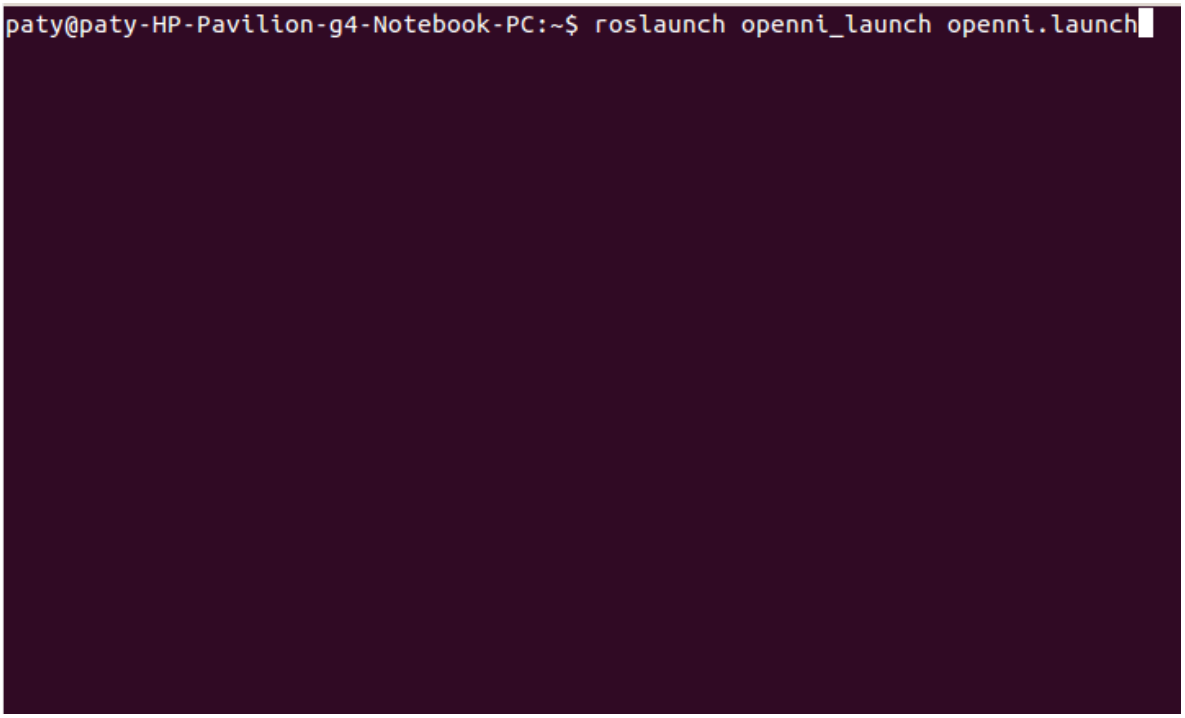


Figura 64. Lanzamiento de OpenNI_Launch.

Una vez realizado el lanzamiento, se podrá registrar la imagen de paridad, así:

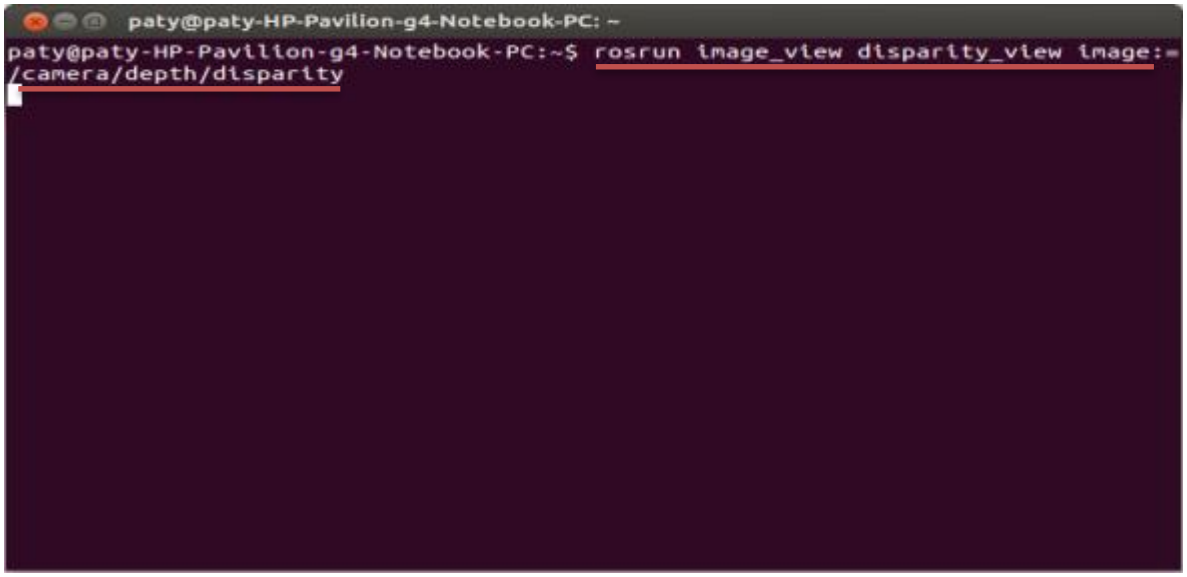


Figura 65. Imagen de paridad.

Para visualizar la imagen en un formato RGB, se procederá a ejecutar el siguiente código:

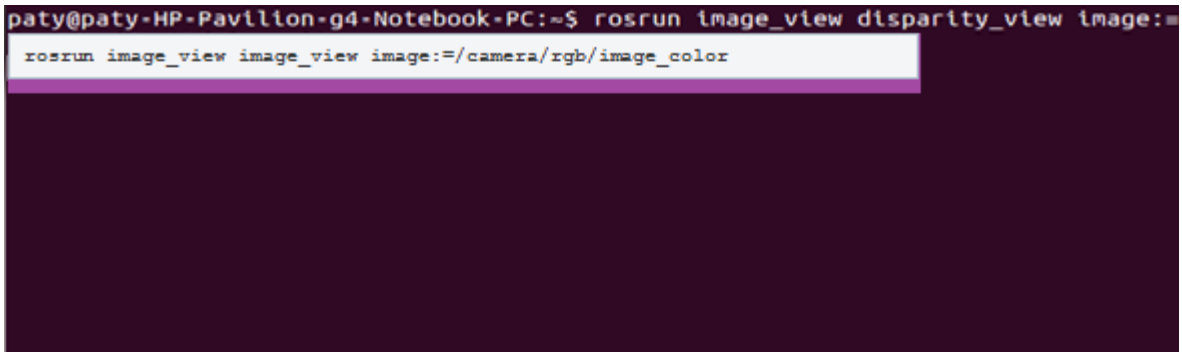


Figura 66. Lanzamiento para imagen RGB.

Con este lanzamiento se podrá ver las imágenes RGB:

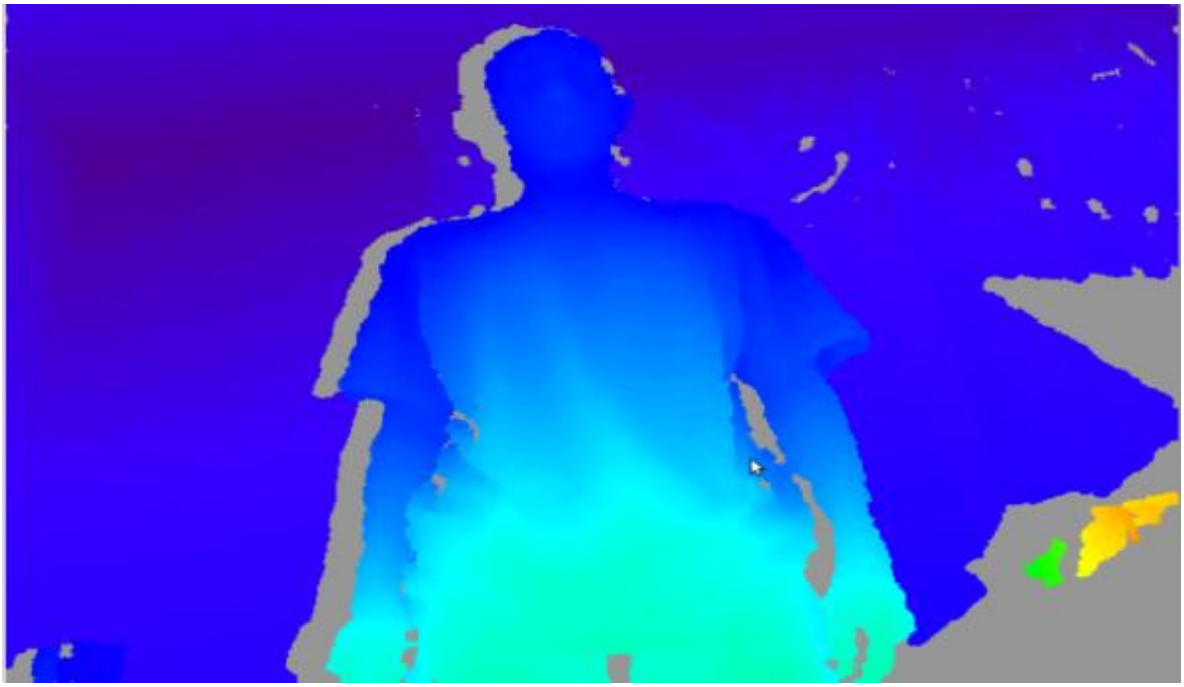


Figura 67. Imágenes obtenidas con Openni camera de Kinect.

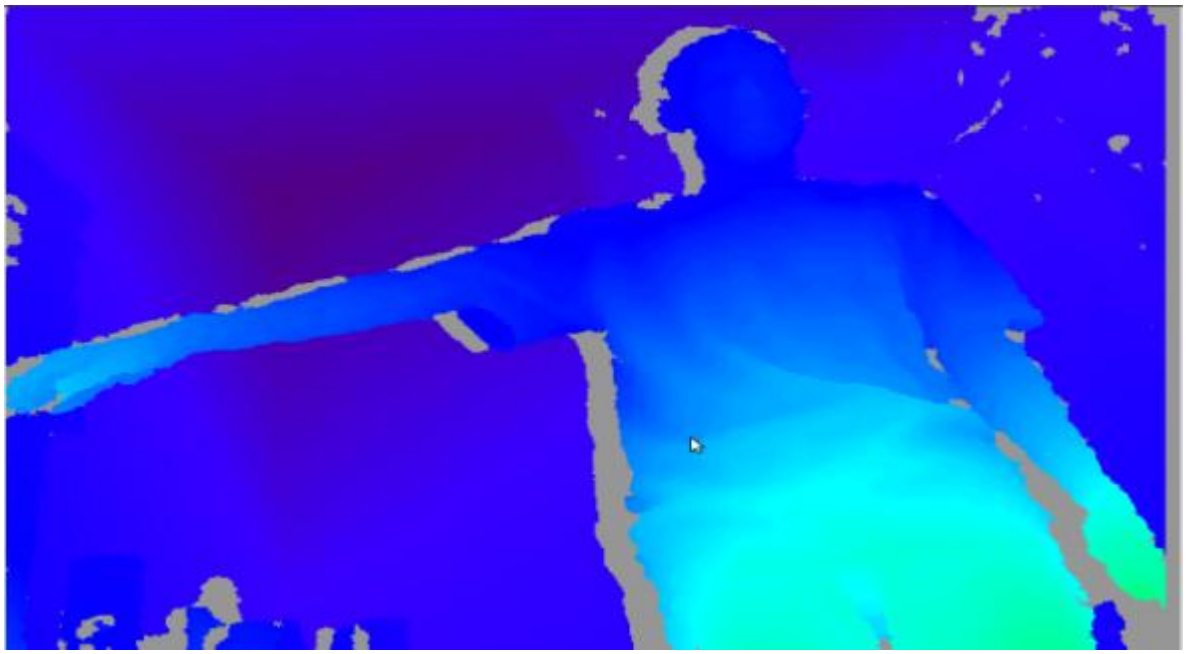


Figura 68. Imágenes obtenidas con Openni de Kinect.

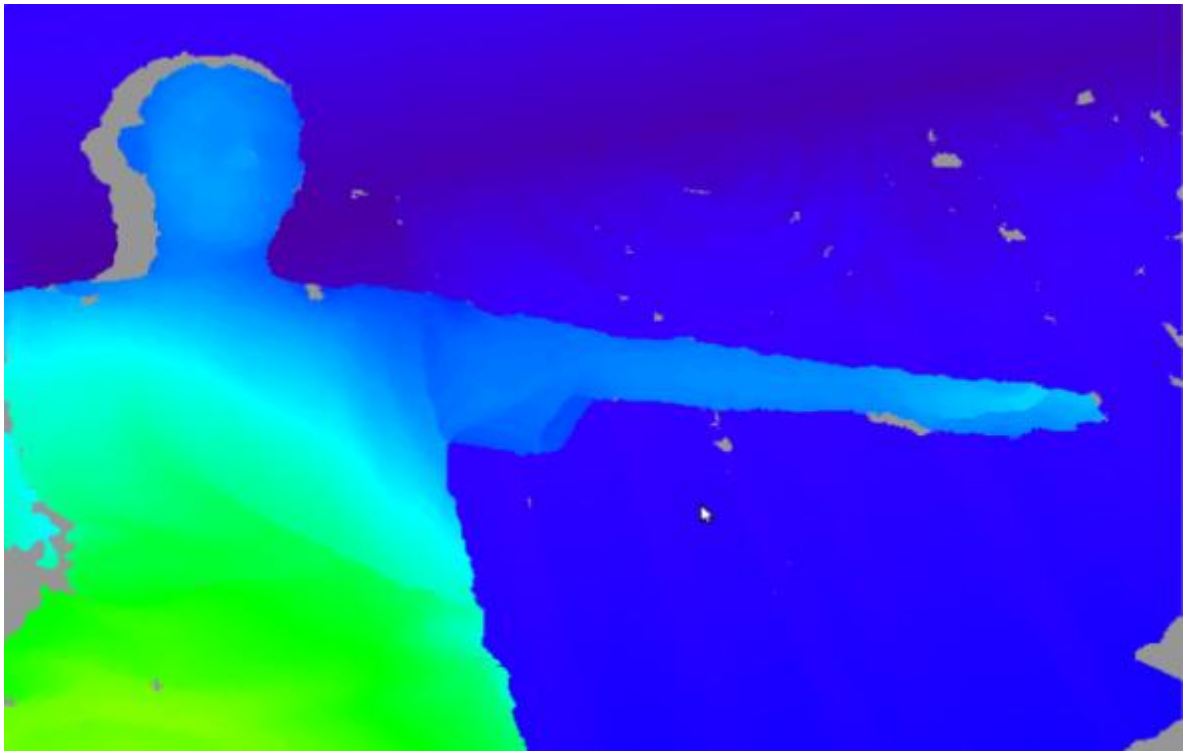


Figura 69. Imágenes obtenidas con Openni de Kinect.

4.3.6.2 OpenNI-Tracker. El OpenNI-Tracker es un paquete de ROS que comprende todo lo relacionado con la lectura del esqueleto y sus articulaciones, para su buen funcionamiento y reconocimiento de Kinect se deberá instalar adicionalmente el controlador NITE.

La instalación del controlador NITE, se deberá realizar por terminal, como lo indica la figura 70.

```
rosrun openni_tracker openni_tracker
```

Figura 70. Código para activar el tracker de Kinect.

Si al llevar a cabo esta instrucción ocurre un error de generación, se deberá descargar el controlador NITE, una vez realizado esto, se procederá a descomprimirlo y guardarlo en la carpeta personal o home de Ubuntu, como lo indica la figura 71.

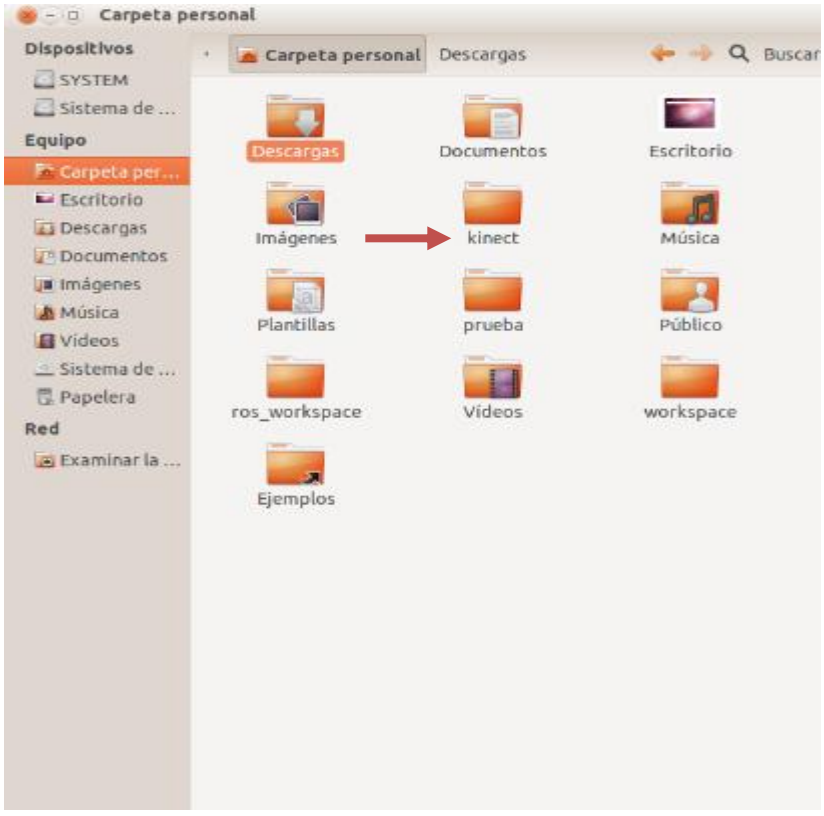


Figura 71. Carpeta creada para guardar el controlador NITE.

Una vez creada la carpeta se deberá guardar el archivo del controlador y descomprimirlo, como se muestra en la figura 72.

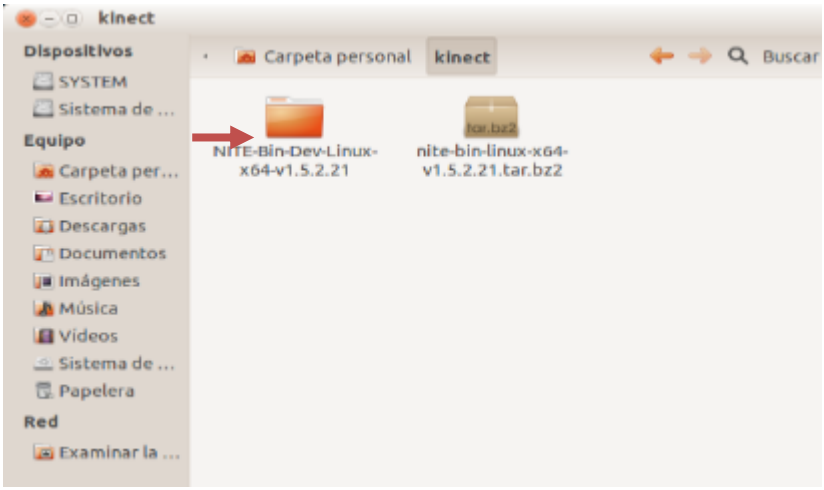


Figura 72. Controlador NITE.

Se deberá acceder a dicha carpeta y realizar la instrucción de instalación, según la figura 73.

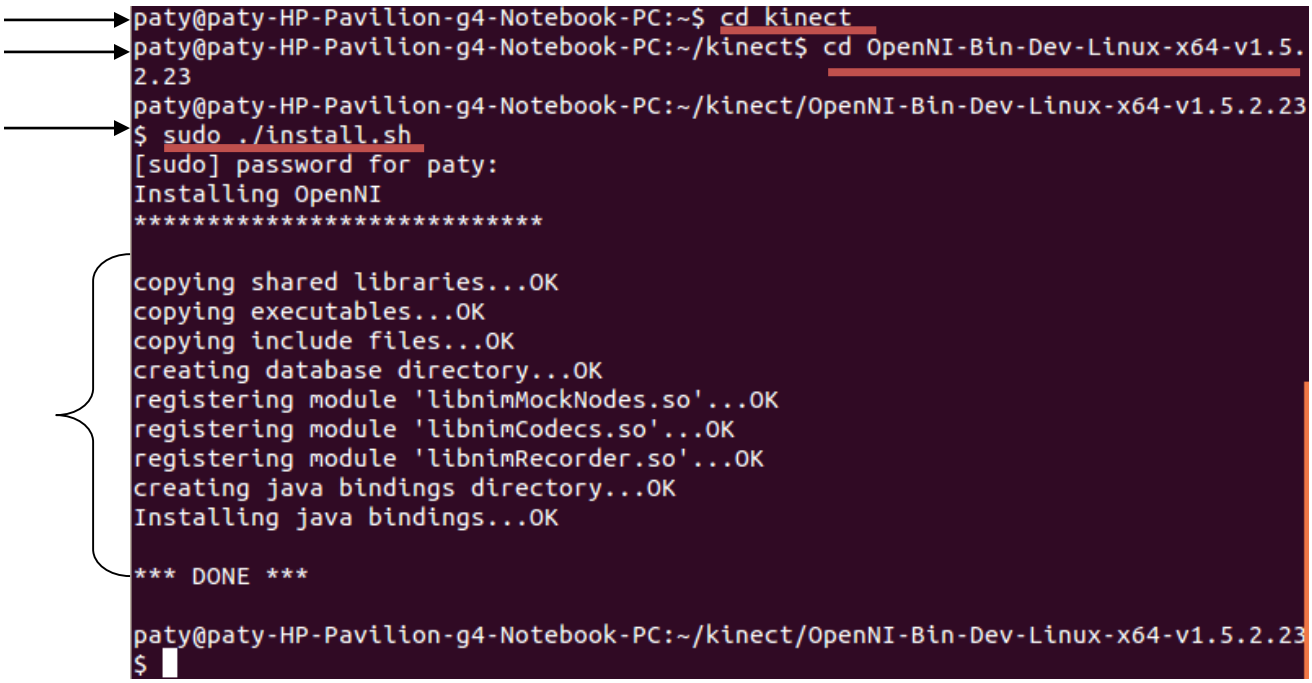


Figura 73. Instalación del Controlador NITE.

Una vez ejecutada la instalación, se deberá verificar el buen funcionamiento del controlador, siguiendo la siguiente ruta: **/NITE-Bin-Dev-Linux-x64-v1.5.2.21/Samples/Bin/x64-Release,** se encontrarán en esta carpeta archivos ejecutables diversos que corresponden a seguimiento del esqueleto y sus partes; para este proyecto se ejecutó el Sample Player, el cual realiza el seguimiento a un jugador y sus articulaciones como se muestra en las siguientes figuras 74,75 y 76.

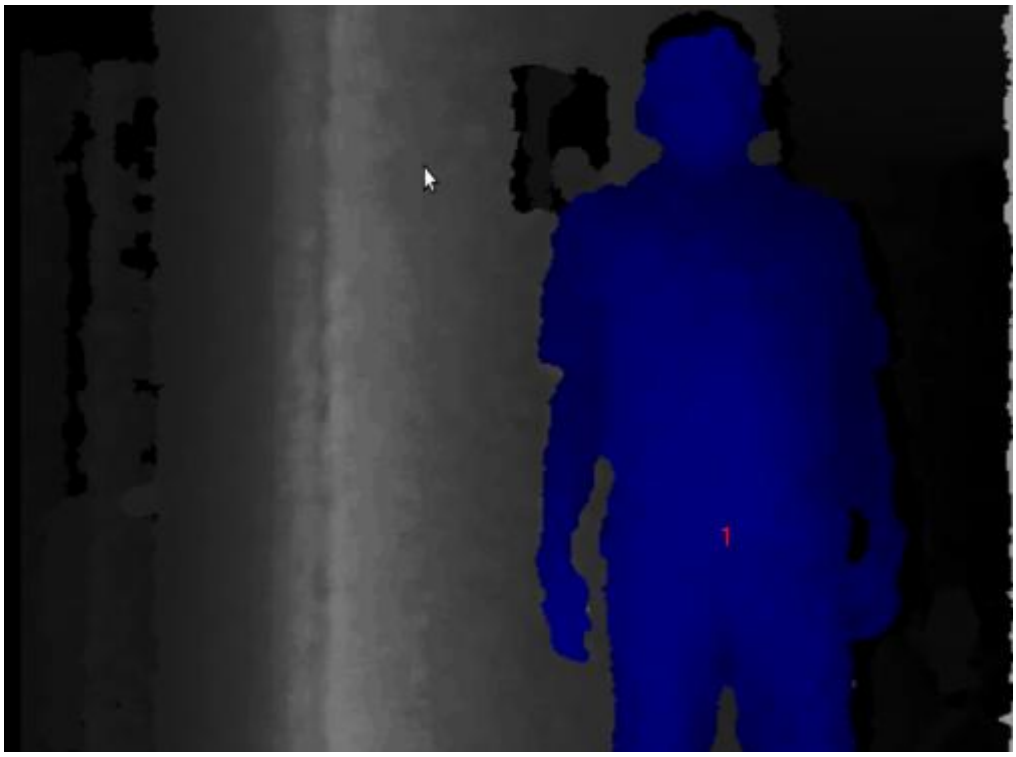


Figura 74. Sample Player NITE.

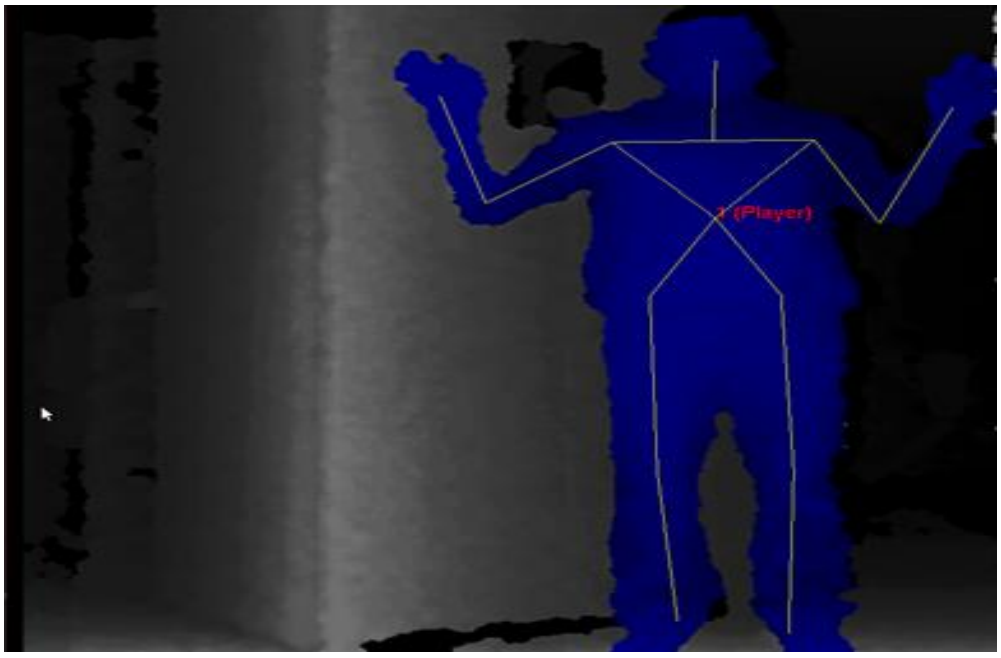


Figura 75. Punto de referencia del jugador.

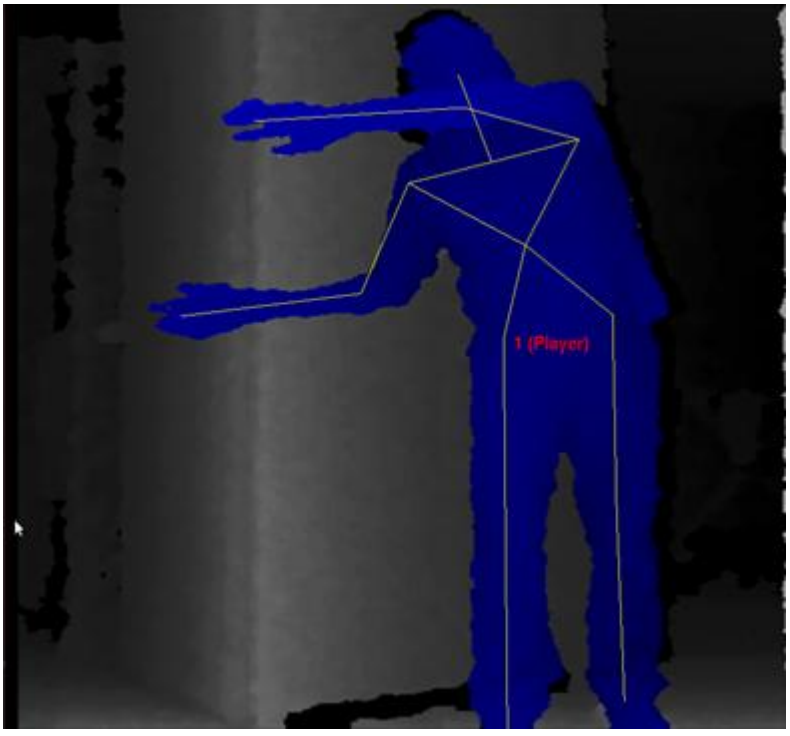


Figura 76. Seguimiento a puntos de articulación.

Ahora se verificará el reconocimiento a dos jugadores en escena, utilizando el controlador NITE de Kinect. Esta aplicación permitirá observar varias figuras en acción, como se indica en las figuras 77 y 78.

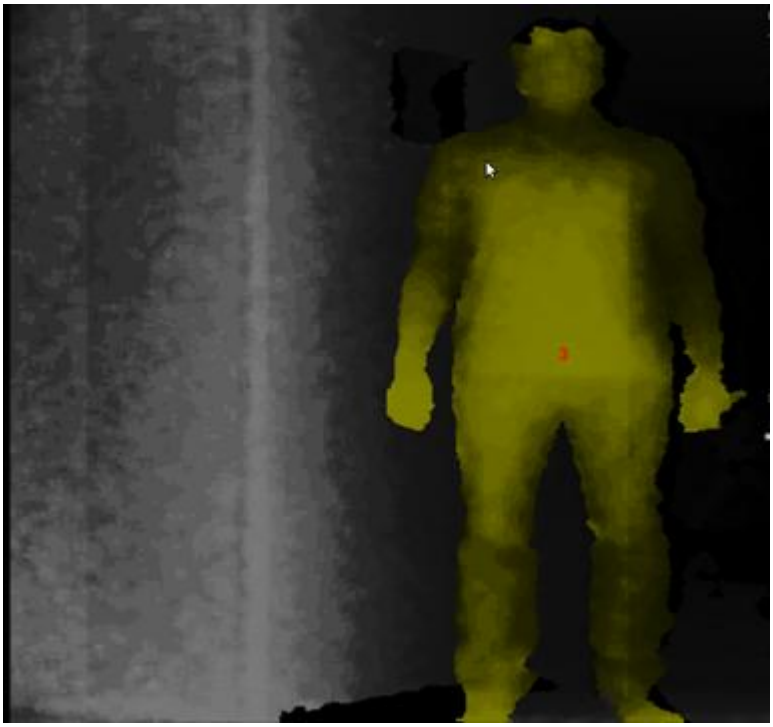


Figura 77. Identificación de un jugador.

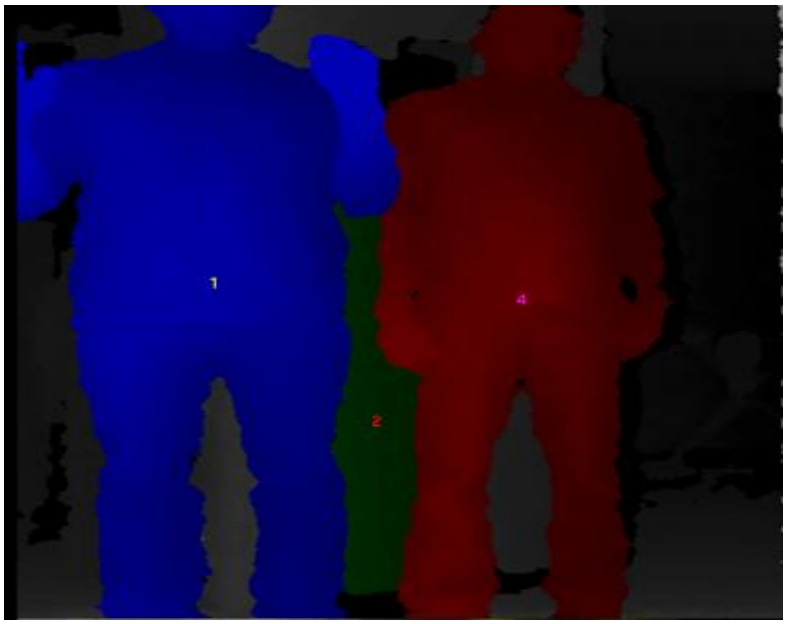


Figura 78. Identificación de varias siluetas en Kinect.

4.3.6.3 Skeleton_Marker. Muestra las articulaciones del esqueleto como marcadores en Rviz. Comprende dos nodos: markers_from_skeleton_msg.py, el cual es el suscriptor del mensaje devuelto por el skeleton_tracker incluido en el paquete y el markers_from_tf.py el cual lee las transformadas directamente desde el OpenNI_Tracker node.msg.

Para su instalación se deberá utilizar la terminal de Ubuntu con el siguiente código como lo muestra la figura 79.

```
paty@paty-HP-Pavilion-g4-Notebook-PC:~$ cd ~/catkin_ws/src
paty@paty-HP-Pavilion-g4-Notebook-PC:~/catkin_ws/src$ git clone https://github.c
om/pirobot/skeleton_markers.git
```

Figura 79. Instalación del paquete Skeleton_Marker.

Se accederá a la carpeta del workspace llamada catkin_ws y allí bajar el paquete para posteriormente ser utilizado. Siguiendo esta ruta ubicamos src y se deberá realizar la siguiente instrucción como lo indica la figura 80.

```
paty@paty-HP-Pavilion-g4-Notebook-PC:~/catkin_ws/src$ cd skeleton_markers
paty@paty-HP-Pavilion-g4-Notebook-PC:~/catkin_ws/src/skeleton_markers$ git check
out hydro-devel
Already on 'hydro-devel'
paty@paty-HP-Pavilion-g4-Notebook-PC:~/catkin_ws/src/skeleton_markers$ cd ~/catk
in_ws
paty@paty-HP-Pavilion-g4-Notebook-PC:~/catkin_ws$ catkin_make
```

Figura 80. Configuración del Skeleton_Marker.

Dentro de src ubicarse en el archivo llamado skeleton_marker y ejecutar el make utilizando catkin, así estará instalado el paquete.

Una vez descargado e instalado en el sistema, se podrá utilizar las siguientes instrucciones para realizar el lanzamiento del mismo en Rviz que es la herramienta visual que tiene por defecto ROS. Así como lo muestra la figura 81.

```
paty@paty-HP-Pavilion-g4-Notebook-PC:~$ roscd skeleton_markers

paty@paty-HP-Pavilion-g4-Notebook-PC:~$ rosrun rviz rviz -d markers.rviz

paty@paty-HP-Pavilion-g4-Notebook-PC:~$ roslaunch skeleton_markers markers.launc
h
```

Figura 81. Lanzamiento del paquete Skeleton_Marker.

El lanzamiento del paquete se muestra en las siguientes figuras:

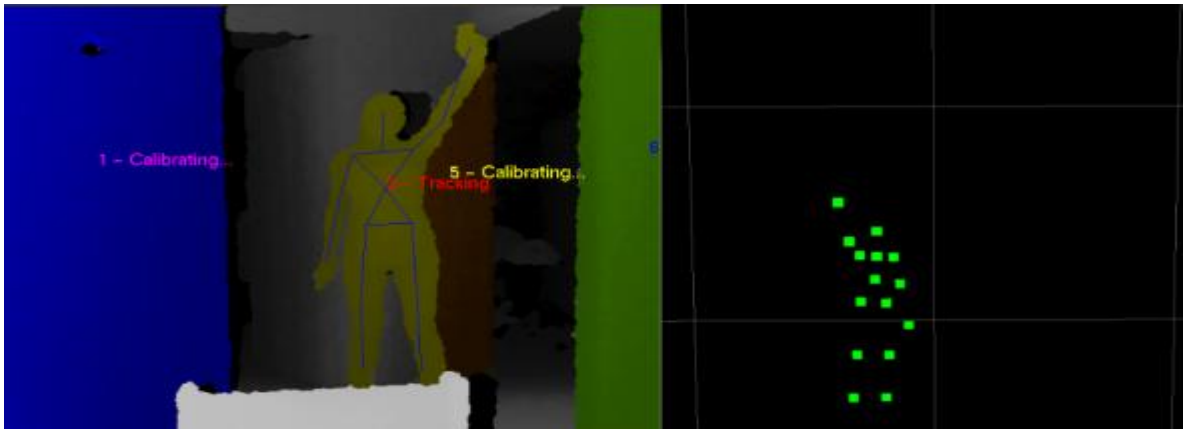


Figura 82. Visualización del Nite y Skeleton_Marker.

En la figura 82 se observa el usuario detectado por el Nite y su figura en rviz con cuadros dinámicos y los puntos de articulación se muestran en la figura 83 y 84.

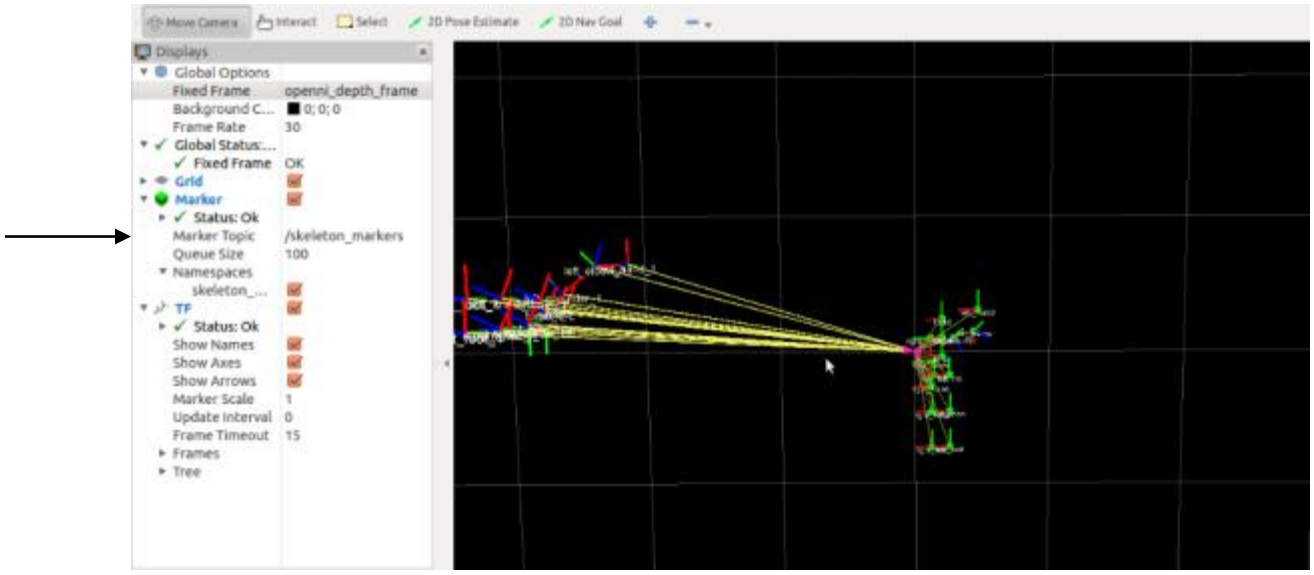


Figura 83. Puntos de esqueletización utilizando Skeleton_Marker.

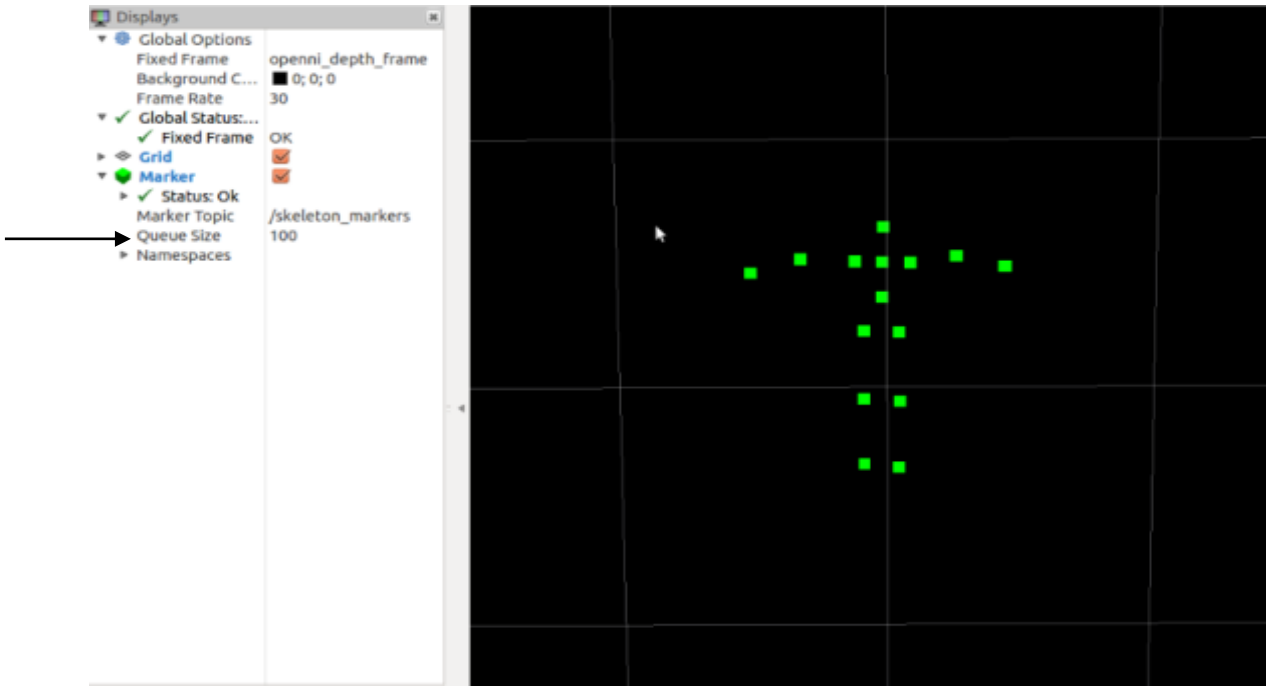


Figura 84. Visualización del Skeleton_Marker.

4.3.7. Configuración de Rviz en ROS. Rviz es un programa de visualización ROS utilizado para ayudar a entender los sistemas de control robótico, proporcionando entornos 2D y 3D para manipulación de imagen, proporcionando a los usuarios otra perspectiva de las imágenes captadas. Se deberá seguir una serie de pasos para cargar el Rviz en ROS:

- En una terminal ejecutar el comando de activación de ROS: **roscore**
- Abrir otro terminal y ejecutar la instrucción para el launch: **roslaunchopenni_launch openni_launch**
- En un tercer terminal abrir el tracker: **rosrun openni_tracker openni_tracker**
- En una última terminal cargar el rviz: **rosrun rviz rviz**

Se obtendrá el siguiente pantallazo:

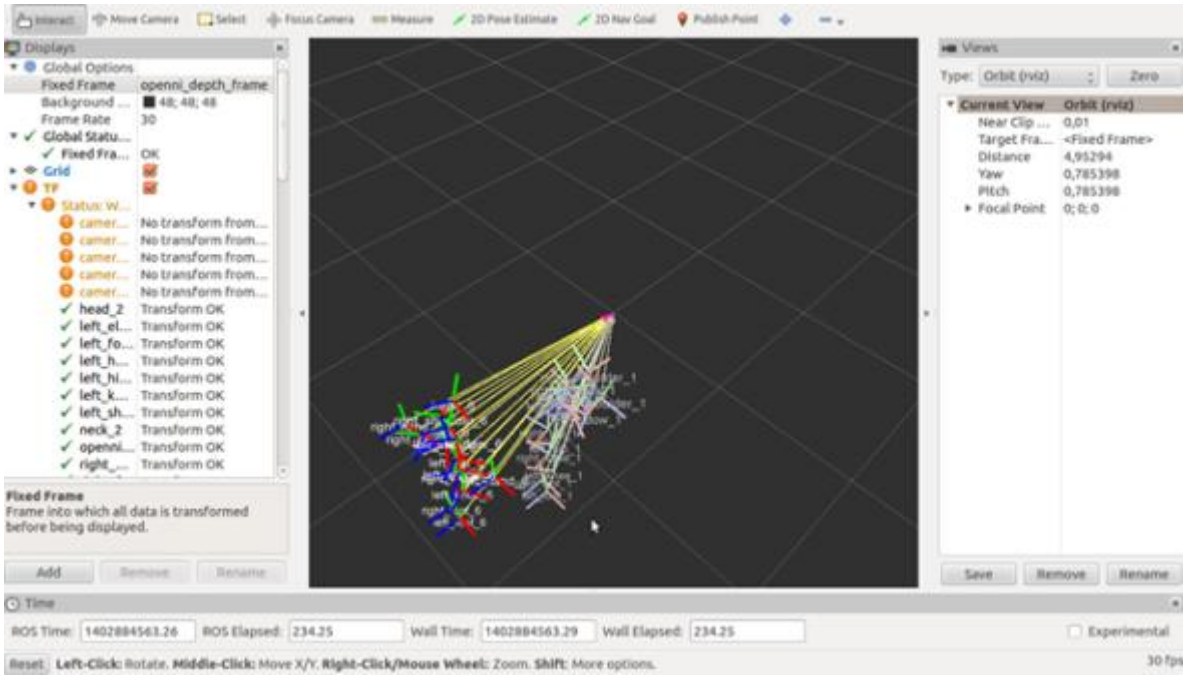


Figura 85. Rviz en ROS.

4.4 PAQUETE OPENNI DE ROS.

A continuación se realizará una breve descripción de las librerías que usa este paquete para su buen funcionamiento.

4.4.1 Librerías.

Librerías necesarias en ROS: Las primeras librerías que se deben declarar son aquellas propias de ROS, como:

- # include <ros/ros.h>: librería de funciones del sistema ROS, es obligatoria como declaración. Ros.h contiene todas las cabeceras necesarias para el uso de las instrucciones más comunes del sistema de ROS, como las referentes al uso de topics y servicios .
- # include <ros/package.h>: Incluye dependencias gráficas como: Vectores, cadenas y mapas. Encierra información de paquetes de ROS [83].

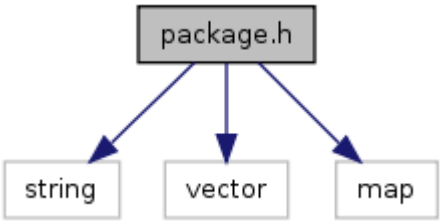


Figura 86. Librería Package de ROS [84].

Archivos que dependen directamente de esta librería:

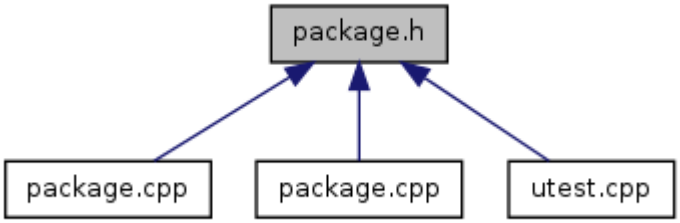


Figura 87. Archivos dependientes de la librería Package [85].

- # include <tf/ transform_broadcaster.h>: Proporciona información sobre un marco de coordenadas, se encarga de toda la administración de los datos provenientes de mensajes.

Librerías propias de OpenNI. Las librerías de OpenNI hacen referencia al funcionamiento del Kinect, los IDS e interfaces.

- # include <xn Codec IDS.h>: Llama los IDS entre ROS y OpenNI.
- # include XnCppWrapper.h>: Los archivos que dependen de esta librería se muestran en la figura 88.

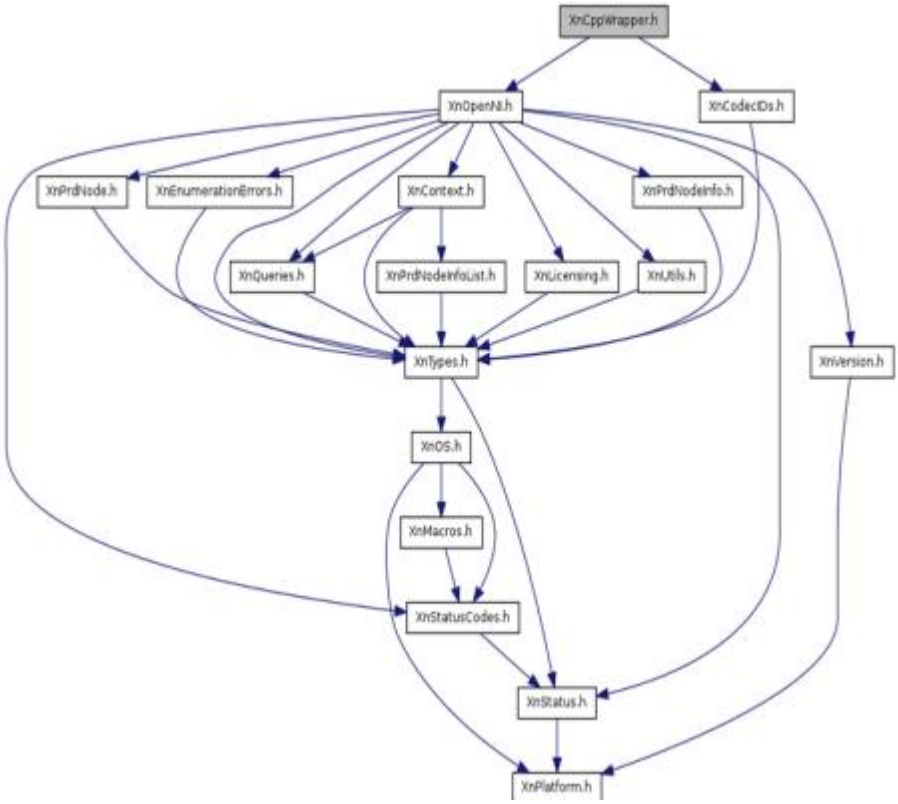


Figura 88. Dependencias de la librería XnCppWrapper.h [86].

Esta librería comprende una interfaz de C++ para OpenNI. Las dependencias directas de esta librería se observan en la figura 89.

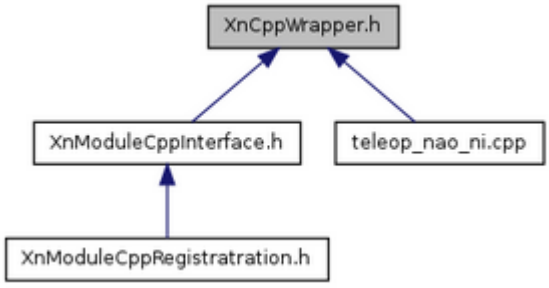


Figura 89. Dependencias directas de la librería XnCppWrapper.h [87].

4.4.1.2. Clases:

class	**xn::AlternativeViewPointCapability**
class	**xn::AudioGenerator**
class	**xn::AudioMetaData**
class	**xn::Capability**
class	**xn::Codec**
class	**xn::Context**
class	**xn::CroppingCapability**
class	**xn::DepthGenerator**
class	**xn::DepthMetaData**
class	**xn::Device**
class	**xn::EnumerationErrors**
class	**xn::ErrorStateCapability**
class	**xn::FrameSyncCapability**
class	**xn::Generator**
struct	**xn::GestureGenerator::GestureCookie**
class	**xn::GestureGenerator**
struct	**xn::HandsGenerator::HandCookie**
class	**xn::HandsGenerator**
class	**xn::ImageGenerator**
class	**xn::ImageMetaData**
class	**xn::IRGenerator**
class	**xn::IRMetaData**
class	**xn::EnumerationErrors::Iterator** An iterator over enumeration errors. More...
class	**xn::NodeInfoList::Iterator** Represents an iterator over a **NodeInfoList** list.
class	**xn::MapGenerator**
class	**xn::MapMetaData**
class	**xn::MirrorCapability**
class	**xn::MockAudioGenerator**
class	**xn::MockDepthGenerator**
class	**xn::MockImageGenerator**
class	**xn::MockIRGenerator**
class	**xn::NodeInfo**
class	**xn::NodeInfoList**
class	**xn::NodeWrapper**
class	**xn::OutputMetaData**
class	**xn::Player**

Clases usadas:

- Xn::Contexto: El contexto es el espacio de trabajo donde la aplicación OpenNI crea el nodo de producción. El contexto se puede entender como la partición del disco, o un directorio, donde OpenNI almacena todo lo que necesita.

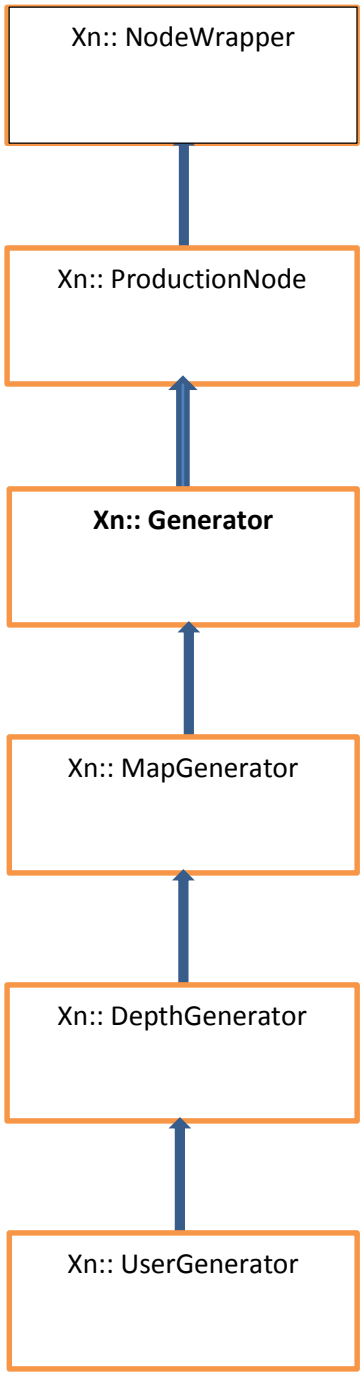


Figura 90. Estructura Jerárquica.

- Xn::UserGenerator: Un nodo de este tipo genera datos relacionados con los usuarios identificados en la escena. Cada usuario se describe independientemente, lo que permite realizar acciones específicas sobre diferentes usuarios. Cada usuario potencial, recibe un identificador (User ID) único. El nodo UserGenerator usa estos identificadores para acceder a datos específicos de cada usuario a través de sus diferentes métodos. Su implementación típica usa la información del mapa de profundidad para extraer información sobre el usuario (usando algoritmo de Shotton). Es por esto que el nodo UserGenerator depende del nodo DepthGenerator, además, se encarga de administrar una serie de funciones de retrollamada (Callback Functions) que se ejecutan asíncronamente como respuesta a los eventos relacionados con los usuarios, entre los eventos más comunes se encuentran:

Detección de nuevo usuario (‘New User’)

Usuario detectado sale de la escena (‘User Exit’)

- Xn::DepthGenerator: Genera un mapa de profundidad a partir de los datos generados por el nodo UserGenerator.
- Xn::SkeletonCapability: Proporciona al nodo UserGenerator la capacidad de seguir la pose de un usuario mediante una representación esqueletal.
- Xn::PoseDetectionCapability: Provee al nodo UserGenerator la capacidad de detectar cuando el usuario se coloca en una cierta pose, que es “Psi”, usuario de frente a la cámara con las manos en alto [88].

Algoritmo de Shotton. Predice con rapidez y precisión posiciones en 3D de las articulaciones del cuerpo a partir de una sola imagen de profundidad. Enfocándose en la captura de movimiento.

Datos de captura de movimiento. El cuerpo humano es capaz de crear una amplia gama de poses que son difíciles de simular, por lo que el algoritmo se basa, en recopilar gran parte de acciones humanas y llevarlas a una base de datos (mocap, captura de movimiento).

Esta base de datos consta de aproximadamente 500 mil frame obtenidas de secuencias de baile, conducción, correr u otros oficios realizados a diario por las personas. Utilizando solo poses estáticas, ya que algunos cambios de frame son tan pequeños que se contarían como insignificantes. Se determina la distancia entre poses así:

P_1 y P_2 están definidas como la máxima distancia $m_j \, ||P_1^j - P_2^j||_2$, la máxima distancia euclidiana entre puntos.

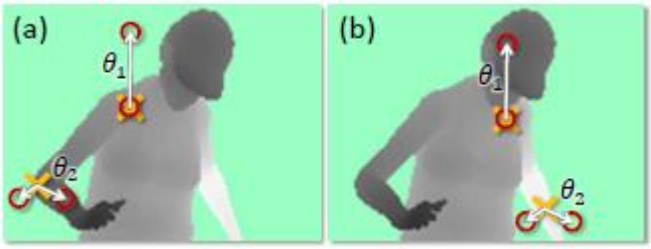


Figura 91. Características de la imagen de profundidad. Las cruces amarillas indican que el pixel ha sido clasificado, y los rojos indican el desplazamiento [89].

Características de la imagen de profundidad. Para caracterizar la imagen de profundidad el algoritmo de Shotton utiliza comparación simple así:

$$f_\theta(I,\mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Ecuación 2. Shotton

Donde $d_I(X)$ representan el modelamiento del punto, y los parametros θ = (u , v) describe el desplazamiento de u y v.

Individualmente estas características proporcionan una señal débil acerca de que parte del cuerpo pertenece el pixel, con el fin de compensar esto, se crea un bosque de decisión (Randomized Decision Forest), para establecer una eficiencia computacional ya que puede leer 3 pixeles de una imagen y realizar 5 operaciones matemáticas.

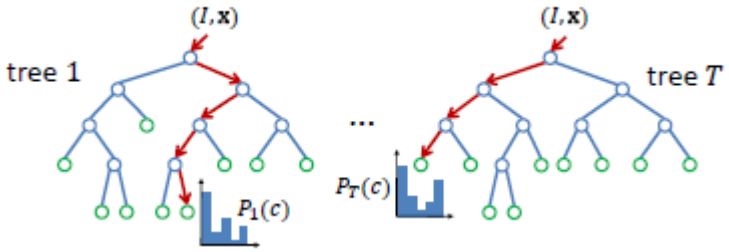


Figura 92. Randomized Decision Forest, Constituye un conjunto de árboles, cada árbol se compone de dos nodos azul que son de división y los verdes que son hoja; Las flechas rojas indican el camino que pueden tomar una entrada en particular [90].

Randomized Decision Forest. Son bosques de decisión T, cada nodo consiste en una función *fθ* y un tiempo T. Para clasificar un pixel en una imagen I, se comienza desde la raíz y se evalúa repetidamente la siguiente ecuación:

$$P(c|I,\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|I,\mathbf{x})$$

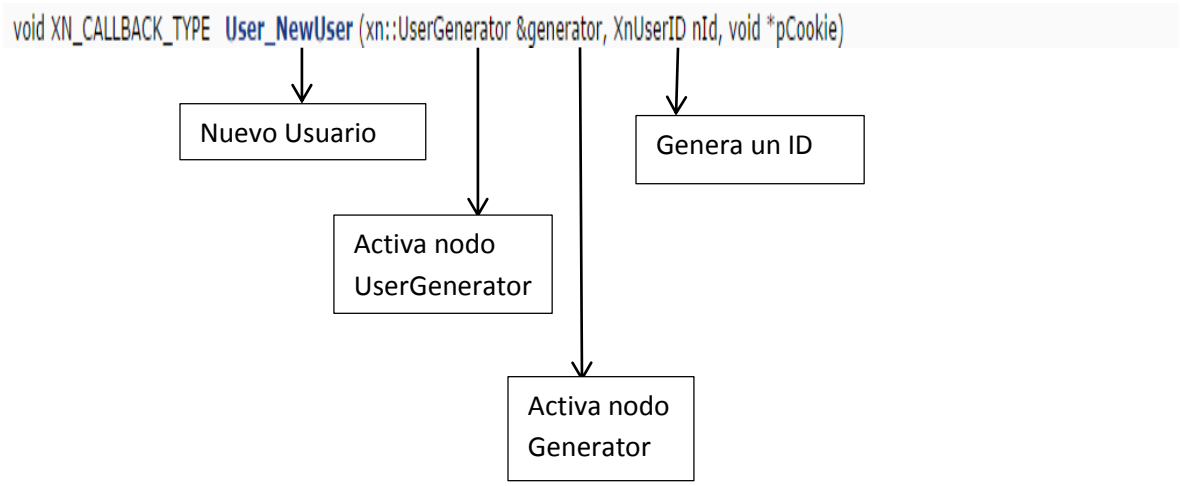
Ecuación 3. Randomized Decision Forest

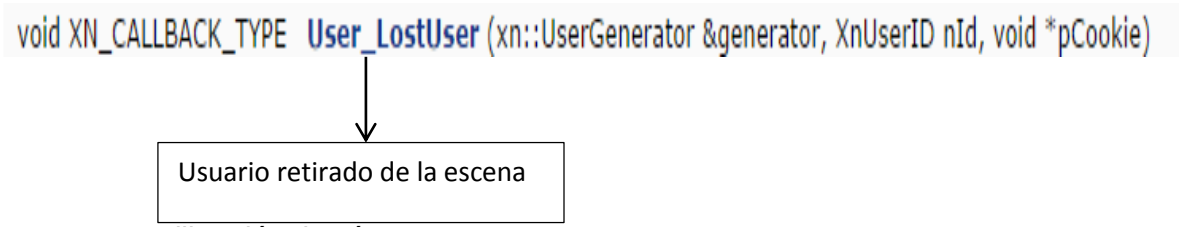
Se realiza la ramificación sea a la izquierda o derecha dependiendo de T, así se ubica en un nodo de hoja generando T en el árbol, y distribuyendo $P_t(c|I,\mathbf{x})$ en el resto del cuerpo.

Cada árbol se entrena con un conjunto diferente de imágenes de síntesis, un subconjunto aleatorio de 2.000 pixeles distribuidos uniformemente a través de partes del cuerpo.

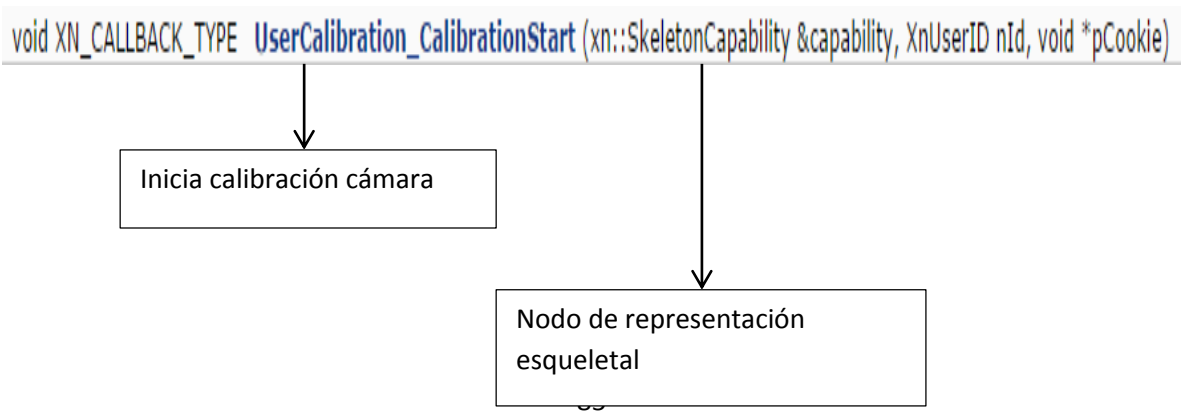
4.4.1.3 Estructura:

- De identificación de usuario





- De calibración de cámara



- De detección de pose

```
void XN_CALLBACK_TYPE UserPose_PoseDetected (xn::PoseDetectionCapability &capability, const XnChar *strPose, XnUserID nId, void *pCookie)
```

4.4.2 Paquete tf (transform). Con este paquete de ROS se puede mantener la relación entre diferentes marcos de referencia usando una estructura en forma de árbol, y que se pueda almacenar a lo largo del tiempo. Las herramientas tf permiten obtener trasformadas (translación y rotación) entre dos marcos de referencia distintos en el tiempo que se requiera.

Tf funciona como un sistema distribuido, por lo que no hay un servidor central de transformadas [91].

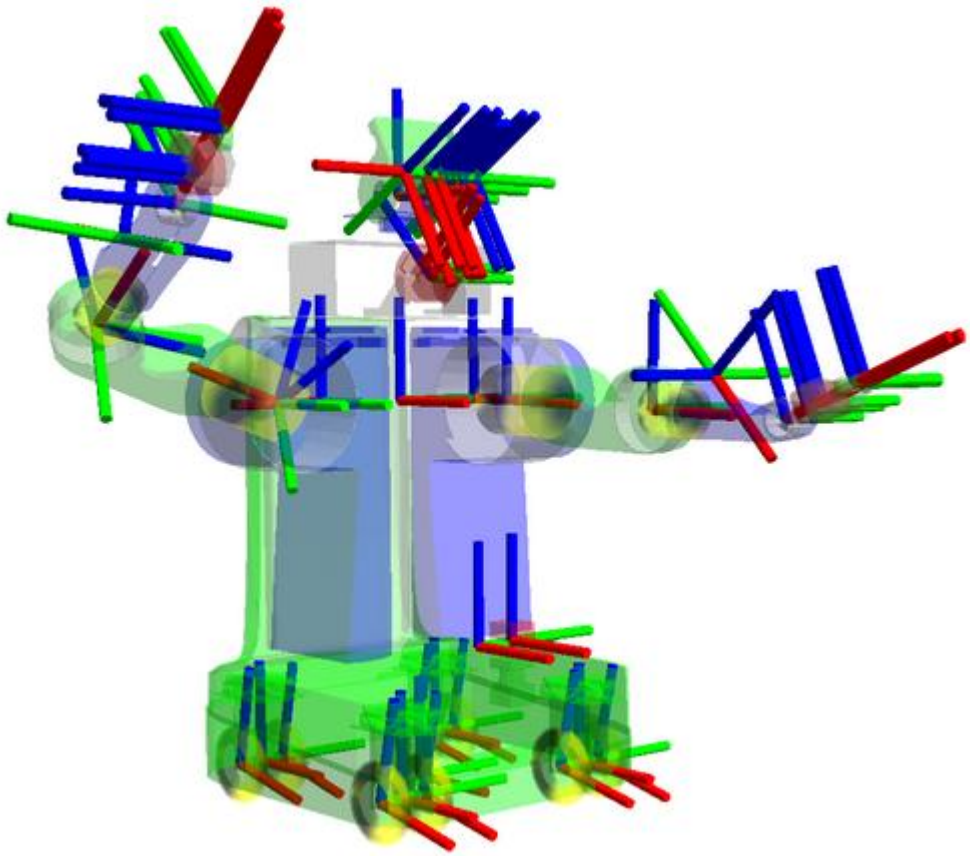


Figura 93. Paquete tf de ROS en un robot [92].

Frame y puntos. Un frame es un sistema de coordenada. El sistema de coordenadas usado en ROS es en 3D, y su referencia son las manecillas del reloj, con X hacia adelante, Y saliendo y Z arriba.

Los frame dentro de un punto se representan usando tf::point; que es equivalente al vector tipo btVector3. Las coordenadas de un punto P en un frame W se escriben como Wp.

Poses. La relación entre dos frame está representada por una pose relativa 6 DOF, una traslación seguida de una rotación, si W y A son dos frame, la pose de A en W está dada por la traslación desde el origen de W's al origen de A's, y la rotación de los ejes coordenados de A's en W.

La traslación es un vector en W's coordenadas, ${}^{W}t_{A.}$ Está representado por tf::Vector3, que es equivalente a el vector btVector3.

La rotación de A esta dada por una matriz de rotación, representada como ${}^{W}_{A}R$, *usando la convención de frame de referencia como un precedente, esto es, la rotación de la trama A en el sistema de coordenadas W. Las columnas de R están formadas a partir de los tres vectores unitarios de los ejes de A en W:* ${}^{W}X_{A}$, ${}^{W}Y_{A}$, *and* ${}^{W}Z_{A.}$

No existe ningún tipo tf para una matriz de rotación, en cambio, tf representa rotaciones tf:: Quaternion, equivalente a btQuaternion .

El tipo quaternion tiene métodos para crear quaterniones de matrices de rotación y viceversa.

Es conveniente describir la traslación + rotación en coordenadas homogéneas, como una única matriz de 4x4 ${}^{W}_{A}T$. *Se puede describir de la siguiente manera: "La pose del frame A con respecto a W". La pose relativa es construida así:*

$$\begin{matrix} {}^{W}_{A}R & {}^{W}t_{A} \\ 0 & 1 \end{matrix}$$

En tf, la pose relativa está representada por tf::pose, que es equivalente al tipo btTransform. Las funciones usadas son: getRotation () o getBasis () para rotación, y getOffset () para traslación.

Frame como puntos de asignaciones. Es una dualidad entre los frame de poses y puntos de mapeo de un cuadro a otro, los pose ${}^{W}_{A}T$ también puede ser leído así: "Transformar un punto del frame A's en W".

Las transformaciones tienen como referencia de código tf::Transform, el cual plantea desplazamiento y transformaciones, las transformaciones se pueden crear utilizando matrices de rotación o cuaterniones para la rotación, y vectores para la traslación [85].

4.5. TRANSFORM FRAME (TF). En un sistema de coordenadas que se utiliza en robotica donde cada coordenada frame tiene exactamente un punto en el origen o desplazado de este, es así como los frame contribuyen a una estructura de árbol llamada "Árbol de Transformación" o "tf tree", semejante a un árbol.

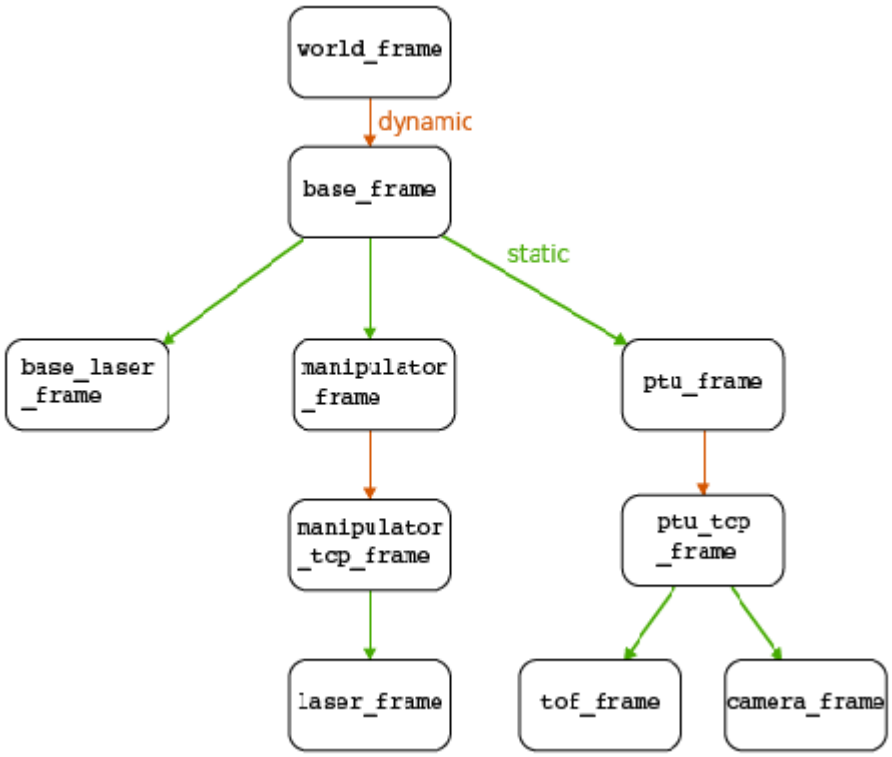


Figura 94. Árbol general tf transform [93].

Como descripción general del árbol tenemos:

El elemento raíz es el worl_frame que especifica el punto de referencia en una palabra, se enlaza de forma dinámica con el frame_base que es la raíz del frame. Si hay una colisión se debe activar el scanner del láser en el brazo que es el laser_frame el cual puede ser transformado en base_frame. Esto incluye todos los ángulos de transformación en la rama desde la base hasta el láser y la transformación es llamada **chain** (tf chain).

Los puntos entre dos frame pueden ser divididos en dos categorías:

- **Frame Estáticos (Static frame)**. Son puntos fijo y no cambian con el tiempo; la calidad de la transformación también es estática y no cambia
- **Frame Dinámica (Dynamic frame).** Posee dos estados fija y en movimiento. Si el frame es fijo la calidad de la transformación solo es afectada por la posición exacta del actuador que este moviendo el frame; si el frame está en movimiento la calidad de transformación es por ejemplo un efecto adicional por la incertidumbre en la velocidad y aceleración.

Diferentes tipos de puntos.

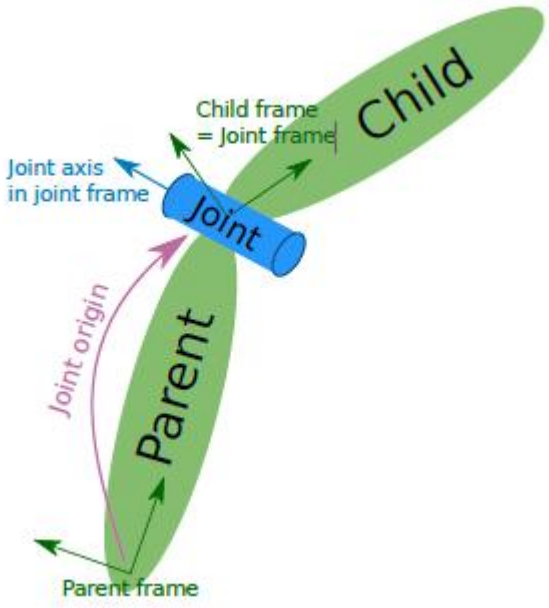


Figura 95. Relación entre frame y puntos [94].

En la figura 95 se observa el parent frame y child frame. Dependiendo del tipo de articulación, esta puede girar alrededor o moverse en dirección de un eje del frame en conjunto (child frame) Para convertir en un valor real la coordenada y la rotación de parent frame debe conocerse child frame, para estas transformaciones se utiliza tf.

Para tf existen diversas formas de clasificar los puntos:

- Revolute joint: Esta articulación tipo bisagra gira alrededor de un eje y tiene un alcance limitado
- Continuous joint: Es la articulación de bisagra continua que gira alrededor de un eje superior sin los limites inferiores
- Prismatic joint: Es un conjunto lineal que se mueve a lo largo de un eje. Tiene un rango limitado
- Planar joint: Esta articulación permite el movimiento en un plano
- Fixed joint: No se considera una articulación debido a los grados de libertad (de traslación y rotación) los cuales están bloqueados.
- Floating joint: No es un conjunto , ya que todos los grados de libertad son gratuitos

4.6 TRANSFORMACIONES. Hay diferentes transformaciones de set de puntos en $R^{3.}$ Que se constituyen en importantes para la robótica y es, la transformación (tf) y la rotación (Cuaterniones). La translación simplemente define un desplazamiento de un objeto dentro de un frame de visualización de acuerdo a la regla:

$$P = (x, y) \rightarrow P' = (x + h, y + k)$$

Ecuación 4. Desplazamiento de un objeto dentro de un frame.

La traslación puede ser escrita en forma de matriz de la siguiente forma:

$$P' = P + T$$

Donde

$$T = \begin{bmatrix} 2.5 \\ 1 \end{bmatrix}$$

Así

$$P' = P + \begin{bmatrix} 2.5 \\ 1 \end{bmatrix}$$

Donde

P = Set de puntos originales

P' = Set de puntos trasladados

Una rotación somete un conjunto de puntos a una rotación alrededor del origen a través de un ángulo. Esto se hace de acuerdo a la regla

$$\begin{aligned} P = (x, y) \quad &\rightarrow \quad P' = (x', y') \\ \text{where} \quad x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta - y\cos\theta \end{aligned}$$

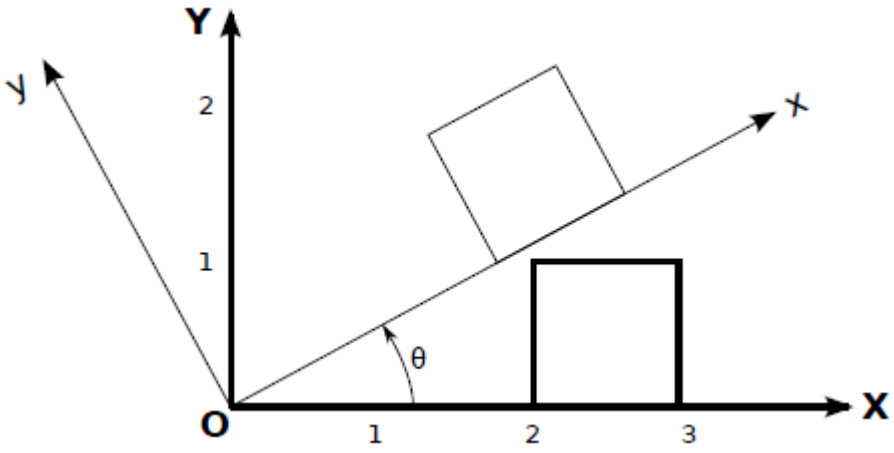


Figura 96. Rotación en R^2 , donde un cuadrado gira alrededor del origen a través de θ [95].

Traslación puede ser descrito como un vector o una matriz si se utiliza coordenadas homogéneas, en contraste la rotación es más complejo describirla.

4.7. ROTACIÓN EN ROS. Para realizar la rotación ROS utiliza los siguientes métodos:

Rotación alrededor de un eje fijo: es un caso especial del movimiento rotacional, de acuerdo al teorema de rotación de Euler, la rotación alrededor de más de un eje al mismo tiempo es imposible, así, si dos rotaciones son forzadas al mismo tiempo en diferente eje, aparecerá un nuevo eje de rotación.

El movimiento de rotación tiene una estrecha relación con el movimiento lineal, el desplazamiento lineal es el producto del desplazamiento angular por el radio del círculo descrito por el movimiento.

$$S= \theta R$$

Ecuación 5. Desplazamiento angular.

La velocidad lineal es el producto de la velocidad angular por el radio del círculo descrito por el movimiento.

$$V=\omega R$$

Ecuación 6. Velocidad angular.

La aceleración tangencial es el producto de la aceleración angular por el radio del círculo descrito por el movimiento.

$$A=\alpha R$$

Ecuación 7. Aceleración Tangencial.

Ángulo de Euler: Los ángulos de Euler fueron introducidos por el famoso matemático Leonard Euler (1707-1783) en su trabajo en la mecánica celeste afirmó y demostró el teorema de que:

“Cualquiera de los dos frame ortonormales independientemente de las coordenadas pueden estar relacionados por una secuencia de rotaciones (no más de tres) sobre ejes coordenados, donde no hay rotaciones sucesivas pueden ser aproximadamente el mismo eje..”

Este teorema indica, que un cuadro se puede girar en otro frame a través de rotaciones sucesivas sobre los ejes coordenados. El ángulo de rotación alrededor de un eje de coordenadas se denomina ángulo de Euler. El teorema de Euler, dice: dos rotaciones sucesivas deben ser sobre diferentes ejes y así permitir la generación de las secuencias de Euler, las cuales son:

xyz	yzx	zxy
xzy	yxz	zyx
xyx	yzy	zxz
xzx	yxy	zyz

Las secuencias se leen de izquierda a derecha. Para la secuencia XYZ significa una rotación alrededor del eje x, seguido por una rotación alrededor del nuevo eje y, seguido por una rotación alrededor del eje z más reciente.

En la secuencia aeroespacial, que es la secuencia ZYX, se puede expresar como un producto de matrices así:

$$\begin{aligned} R &= R_{\phi}^{x} R_{\theta}^{y} R_{\psi}^{z} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\phi & sin\,\phi \\ 0 & -sin\,\phi & cos\,\phi \end{bmatrix} \begin{bmatrix} cos\,\theta & 0 & -sin\,\theta \\ 0 & 1 & 0 \\ sin\,\theta & 0 & cos\,\theta \end{bmatrix} \begin{bmatrix} cos\,\psi & sin\,\psi & 0 \\ -sin\,\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

En la ecuación de estados:

$$R = R_{\phi}^{x} R_{\theta}^{y} R_{\psi}^{z}$$

La rotación del ángulo de encabezamiento alrededor del eje z, seguido por una rotación a través del ángulo de elevación ψ sobre el nuevo eje y, la rotación final es a través del ángulo Φ en el nuevo eje x.

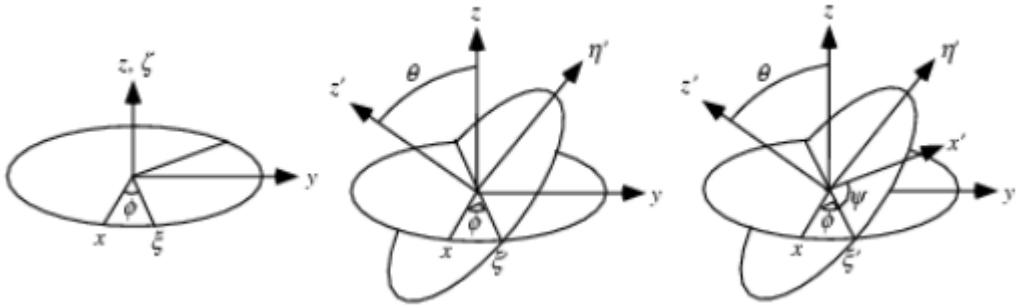


Figura 97. Ángulos de Euler [96].

$$A = \begin{pmatrix} \cos\phi\cos\psi - \sin\phi\cos\theta\sin\psi & \sin\phi\cos\psi + \cos\phi\cos\theta\sin\psi & \sin\theta\sin\psi \\ -\cos\phi\sin\psi - \sin\phi\cos\theta\cos\psi & -\sin\phi\sin\psi + \cos\phi\cos\theta\cos\psi & \sin\theta\cos\psi \\ \sin\phi\sin\theta & -\cos\phi\sin\theta & \cos\theta \end{pmatrix}$$

Figura 98. Matriz de transformación [97]

Es una ventaja tomar como fundamento del sistema del cuerpo los ejes principales, o sea, para los cuales el tensor de inercia es diagonal. Adicionalmente, el centro de simetría es uno de los ejes principales y será escogido como sistema de coordenadas fijo en dirección z al cuerpo. Para el movimiento rotacional sobre un punto fijo, el uso de la segunda ley de Newton nos lleva a un conjunto de ecuaciones conocidas como las ecuaciones del movimiento de Euler.

$$\left(\frac{d\mathbf{L}}{dt}\right)_s = \left(\frac{d\mathbf{L}}{dt}\right)_b + \omega \times \mathbf{L} \quad \Longrightarrow \quad \begin{aligned} I_1\dot{\omega}_1 - \omega_2\omega_3(I_2 - I_3) &= N_1 \\ I_2\dot{\omega}_2 - \omega_3\omega_1(I_3 - I_1) &= N_2 \\ I_3\dot{\omega}_3 - \omega_1\omega_2(I_1 - I_2) &= N_3 \end{aligned}$$

Ecuación 8. Movimiento de Euler.

Con los ω_i y N_i, $1 \leq i \leq 3$, medidos en el sistema del cuerpo. Se puede deducir que el cambio en los ángulos de Euler, está dado por:

$$\dot{\phi} = -\omega_x^s \frac{\sin\phi\cos\theta}{\sin\theta} + \omega_y^s \frac{\cos\phi\cos\theta}{\sin\theta} + \omega_z^s \qquad \begin{aligned} \dot{\theta} &= \omega_x^s\cos\phi + \omega_y^s\sin\phi \\ \dot{\psi} &= \omega_x^s\frac{\sin\phi}{\sin\theta} - \omega_y^s\frac{\cos\phi}{\sin\theta} \end{aligned}$$

Ecuación 9. Ángulos de Euler.

Donde s se refiere a las velocidades angulares medidas en el sistema del espacio.

Cuaterniones: Los cuaterniones fueron introducidos por W.R. Hamilton en 1843, después de que él y otros matemáticos como Wessel, Gauss, Argand, Mourey y Servois buscaron por muchos años un sistema numérico que describiera puntos del espacio tridimensional en forma similar a como los números complejos describen los puntos del plano, así:

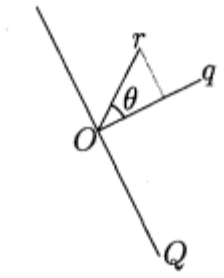


Figura 99. Los vectores (q1,q2,q3) y (x,y,z) están en el plano de la figura y el plano Q es norma a éste [98].

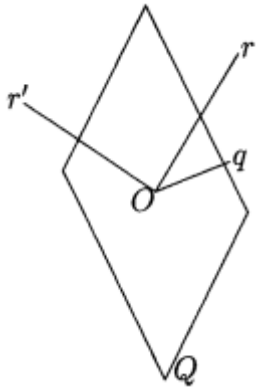


Figura 100. Representación geométrica de los cuaterniones q, r y r' [99]

Los cuaterniones son números hipercomplejos de la forma a + bi + cj dk, donde a, b, c, d son números reales y las tres unidades imaginarias i, j, k tienen cuadro igual a -1:

$$\mathbf{l^2 = j^2 = k^2 = -1}$$

Y además,

$$\mathbf{lj=k=-ji;} \qquad \mathbf{jk=i=-kj;} \qquad \mathbf{ki=j=-ki}$$

Por lo que la multiplicación de cuaterniones no es conmutativa, pero es asociativa y distributiva sobre la suma. En el desarrollo de las matemáticas, los cuaterniones tuvieron

una gran importancia, siendo el primer ejemplo de campo no conmutativo y la base del algebra vectorial que se emplea actualmente (en las que las unidades imaginarias i, j, k pasaron a ser vectores **i, j, k**)

Rotación en cuaterniones. Es conveniente identificar cada punto (a_1, a_2, a_3) ϵ R^3 con el vector que une el origen con dicho punto, de tal manera se puede decir que indistintamente de dos puntos o de vectores. Además, si (a_1, a_2, a_3) y (b_1, b_2, b_3) son dos vectores cualesquiera y θ es el ángulo entre ellos, entonces

$$a_1b_1 + a_2b_2 + a_3b_3 = \sqrt{a_1^2 + a_2^2 + a_3^2}\,\sqrt{b_1^2 + b_2^2 + b_3^2}\,\cos\theta.$$

Ecuación 10. Rotación en cuaterniones.

(El lado izquierdo de esta ecuación es el producto escalar de los vectores (a_1, a_2, a_3) y (b_1, b_2, b_3), mientras que, $\sqrt{a_1^2 + a_2^2 + a_3^2}$ es la longitud del vector (a_1, a_2, a_3)). Por otra parte, se cumple que

$$(a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_3) = \sqrt{a_1^2 + a_2^2 + a_3^2}\,\sqrt{b_1^2 + b_2^2 + b_3^2}\,\operatorname{sen}\theta\,(c_1, c_2, c_3),$$

Ecuación 11. Producto escalar y longitud del vector.

Donde (c_1, c_2, c_3) es un vector de norma igual a 1 ortogonal a (a_1, a_2, a_3) y (b_1, b_2, b_3). El sentido del vector (c_1, c_2, c_3) coincide con el dado por la regla de la mano derecha.

Nociones básicas en cuaterniones.

Conjugado del cuaternion. Sea **q= a + bi + cj +dk**, que denotaremos por q*, se define por **q*= a - bi – cj –dk.** A partir de:

$$\mathbf{i^2 = j^2 = k^2 = -1}$$

ij=k=-ji; jk=i=-kj; ki=j=-ki

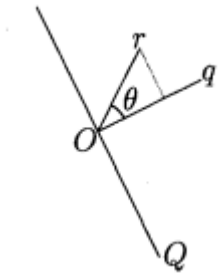
Se tiene entonces que **(qq´)* = q´q***, para cualquier par de cuaterniones q, q´y que qq*es un número real no negativo. Denotaremos por |q| la norma de q, la cual definimos por **|q| =** √**qq***, luego

$$|q| = \sqrt{a^2 + b^2 + c^2 + d^2}$$

Ecuación 12. Norma de q.

Un cuaternion r es un cuaternion puro si su parte real vale cero, lo cual equivale a que r sea de la forma **r = xi + ji + zk,** el cuaternion puro, **r = xi + ji + zk,** puede identificarse con el punto (x, y, z) de $R^{3.}$

Sea Q el plano en R^3 que pasa por el origen y es norma a ($q_{1,}$ q_2, q_3), así:

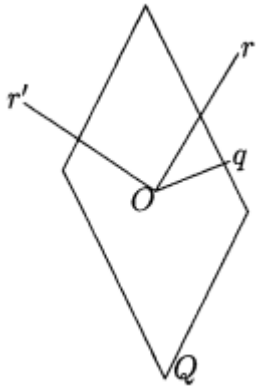


Se puede observar que |r| cos θ es la proyección de (x, y, z) en la dirección ($q_{1,}$ q_2, q_3), por lo que

$$r' \equiv -qrq^* = r - 2|r| \cos\theta \, q$$

Ecuación 13. r' y r.

Representa la imagen de r bajo la reflexión en el plano Q, así:



Identificando los cuaterniones puros con puntos de R^3 , la aplicación

$$r \mapsto -qrq^*$$

Representa la reflexión en el plano normal a q que pasa por el origen, si |q| = 1.

Consideremos ahora dos cuaterniones puros de norma unidad, q y p.

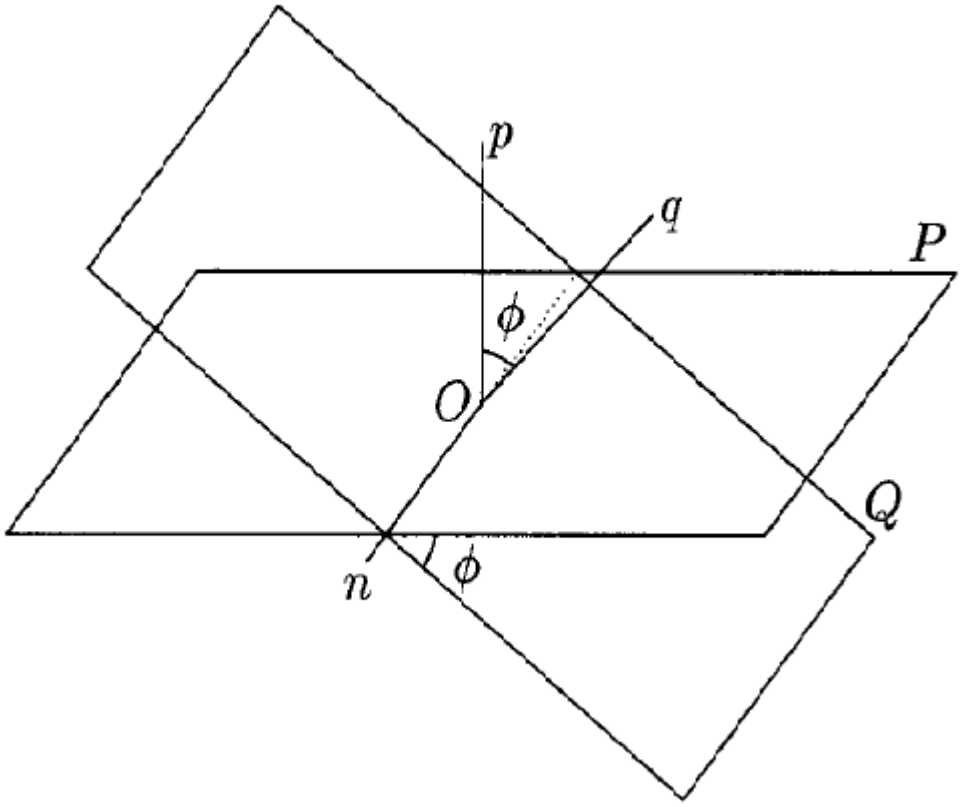


Figura 101. Los cuaterniones puros q, p y n tienen normal igual a 1 y la dirección de n, siendo ortogonal a las de q y p, está en la línea de intersección de los planos Q y P. Las direcciones de q y p se han escogido de tal manera que Φ es menor que 90°. [100].

La imagen de r bajo la reflexión en el plano Q, normal a q, seguida de la reflexión en el plano P, normal a p.

Escribiendo **$p = p_1 i + p_2 j + p_3 k$**, se define:

$$\begin{aligned} pq &= (p_1 i + p_2 j + p_3 k)(q_1 i + q_2 j + q_3 k) \\ &= -(p_1 q_1 + p_2 q_2 + p_3 q_3) - (q_2 p_3 - q_3 p_2) i - (q_3 p_1 - q_1 p_3) j - \\ &\qquad - (q_1 p_2 - q_2 p_1) k \\ &= -\cos\phi - \operatorname{sen}\phi\, n, \end{aligned}$$

Donde n es un cuaternion puro de norma unidad que representa una dirección normal a las direcciones de q y p, con el sentido dado por la regla de la mano derecha y Φ es el ángulo entre las direcciones de q y p. Por lo tanto, la dirección de n está a lo largo de la intersección de los plano Q y P, y Φ es uno de los ángulos formados entre dichos planos, se nota que , mientras que q y p son cuaterniones puros, en general, pq no lo son, sin embargo, |pq| = 1.

La aplicación:

$$r \mapsto (pq)r(pq)$$

Es el resultado de dos reflexiones, representa una rotación en R^3 alrededor del origen. El eje de dicha rotación está en la dirección de n, lo cual se puede deducir notando que n queda invariante bajo esta aplicación por estar en ambos planos, Q y P, y los puntos de cada plano son invariantes bajo la reflexión en ese plano. Alternativamente, dado que n es un cuaternión puro, **n* = -n**, así que **nn* = 1** equivale a $\mathbf{n^2 = -1}$, por lo que bajo la composición de reflexiones en los planos Q y P.

$$n \mapsto (pq)n(pq)^* = (\cos\phi + \operatorname{sen}\phi\, n)n(\cos\phi - \operatorname{sen}\phi\, n) = (\cos\phi\, n - \operatorname{sen}\phi)(\cos\phi - \operatorname{sen}\phi\, n) = n.$$

Según lo anterior, se puede deducir de los cuaterniones lo siguiente:

- Cualquier rotación en R^3 alrededor del origen equivale a la composición de dos reflexiones sobre los planos que pasan por el origen. Ambos planos contienen al eje de rotación y el ángulo entre sus normales es la mitad del ángulo de rotación
- La composición de cualquier número de rotaciones en R^3 alrededor del origen equivale a una sola rotación alrededor de algún eje que pasa por el origen. Esta afirmación es conocida como Teorema de Euler, expone de que la composición de cualquier número de rotaciones alrededor del origen corresponde a un producto de cuaterniones de norma igual a 1, el cual es también es cuaternion de norma igual a 1, que es de la forma **cos (α/2)+ sen (α/2)n**, donde n es un cuaternion puro de norma igual a 1.
- La reflexión sobre cualquier plano que pase por el origen equivale a una rotación por 180° alrededor de la dirección normal al plano seguida de la inversión a través del origen ((x, y, z)--→ (-x, -y –z)). En efecto, si n es un cuaternion puro de norma igual a 1, normal a un plano Q, la rotación por π radianes alrededor de n está representada por el cuaternion **cos(π/2)+ sen(π/2)n = n**, por lo que el efecto de esta rotación seguida de la inversión a través del origen está dada por **r-→-(nrn*)**
- El grupo de rotaciones en R^3 alrededor del origen puede identificarse con el espacio real proyectivo tridimensional RP^3 (es decir, son topológicamente

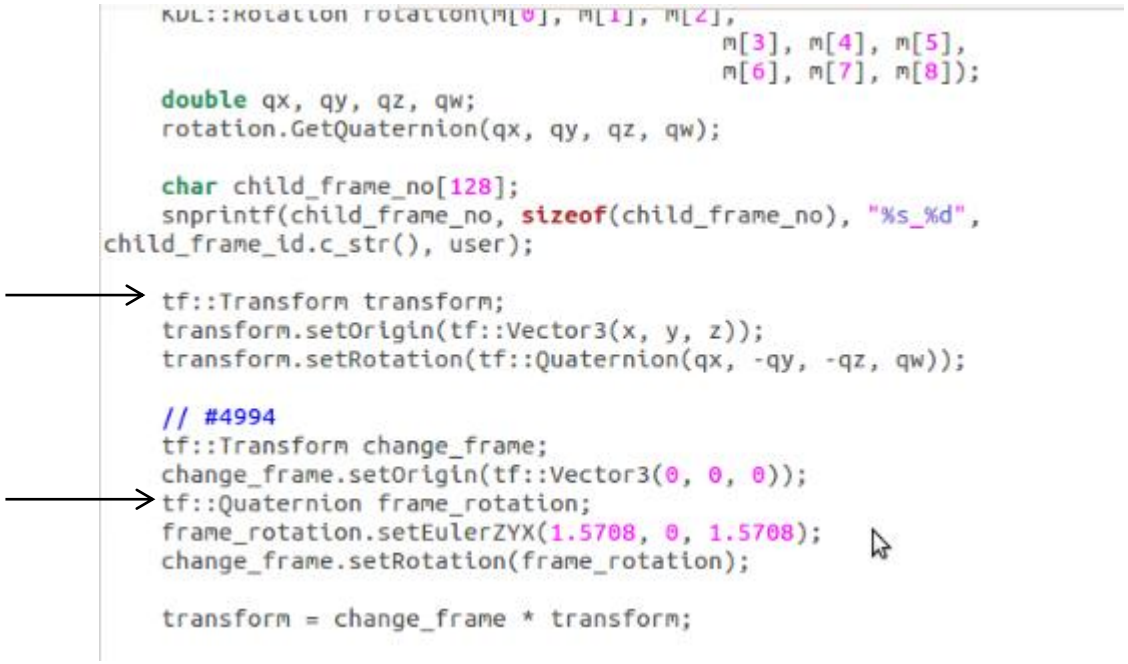
equivalentes). Este hecho sigue de que los cuaterniones de norma igual a 1 están en correspondencia uno a uno con los puntos de la esfera S^3 en R^4 y de que los cuaterniones de norma igual a 1, q y –q, que corresponden a puntos de S^3 diametralmente opuestos, representan una misma rotación en $R^{3.}$ El espacio proyectivo RP^3 se define como el conjunto de rectas en R^4 que pasan por el origen. Cada una de tales rectas intersecta la esfera de radio 1, $S^{3,}$ en dos puntos diametralmente opuestos de S^3 y las rectas que pasan por el origen [101].

4.8 COMPARACIÓN ENTRE LOS MÉTODOS DE ROTACIÓN. En robótica los métodos de rotación son de alta importancia dependiendo de la aplicación que se requiera, se presenta una breve comparación entre métodos de rotación presentando ventajas y desventajas de los mismos:

Propiedad	**Matriz**	**Ángulos de Euler**	**Cuaternion**
Rotación de puntos entre espacios coordenados	Posible	Imposible	Imposible
Concatenación o incremento rotacional	Posible pero toma más tiempo que cuaternion	Imposible	Posible y más rápido que la forma matricial
Interpolación	básicamente Imposible	Posible, pero causa aliasing y bloqueo de ejes	Proporciona interpolación suave
Interpretación humana	Dificultad	Fácil	Dificultad
Almacenamiento en memoria	Nueve números	tres números	Cuatro números
Representación y orientación	Si	no-número infinito de ángulos de Euler	Exactamente dos distintas representaciones para cualquier orientación
Posible llegar a ser invalida	Puede ser invalida	tres números forman una orientación valida	Puede ser invalida

Tabla 2. Comparación entre los métodos de rotación.

La aplicación de los cuaterniones y del paquete tf (transform frame) en el paquete OpenNI se muestran a continuación:



```
    KDL::Rotation rotation(m[0], m[1], m[2],
                           m[3], m[4], m[5],
                           m[6], m[7], m[8]);
    double qx, qy, qz, qw;
    rotation.GetQuaternion(qx, qy, qz, qw);

    char child_frame_no[128];
    snprintf(child_frame_no, sizeof(child_frame_no), "%s_%d",
child_frame_id.c_str(), user);

    tf::Transform transform;
    transform.setOrigin(tf::Vector3(x, y, z));
    transform.setRotation(tf::Quaternion(qx, -qy, -qz, qw));

    // #4994
    tf::Transform change_frame;
    change_frame.setOrigin(tf::Vector3(0, 0, 0));
    tf::Quaternion frame_rotation;
    frame_rotation.setEulerZYX(1.5708, 0, 1.5708);
    change_frame.setRotation(frame_rotation);

    transform = change_frame * transform;
```

Figura 102. TF y Cuaternion en paquete OpenNI.

5. MEDIDAS

Para la toma de medidas se utilizará el paquete Skeleton_Marker de ROS, el cual muestra los ejes coordenados X, Y, Z en una interfaz propia del software llamada RViz y la imagen de profundidad con esqueletizacion del Nite; se proyectará la imagen captada por la cámara RGB del Kinect y las articulaciones del usuario.

Para la obtención de valores confiables, se deberá establecer el punto cero del usuario frente a la cámara y tomar esa distancia como base para desarrollar las mediciones siguientes; así como los registros que generen un error en el paquete ya sea en las manos, los pies u otras articulaciones necesarias para establecer el esqueleto y sus valores.

5.1 POSICIÓN INCORRECTA DE ARTICULACIONES. Utilizando el paquete OpenNI_Tracker de ROS, se encontró que algunas posiciones del usuario generan un error en la medida, lo que significa que para una plataforma robótica este valor no tendría la confiabilidad esperada.

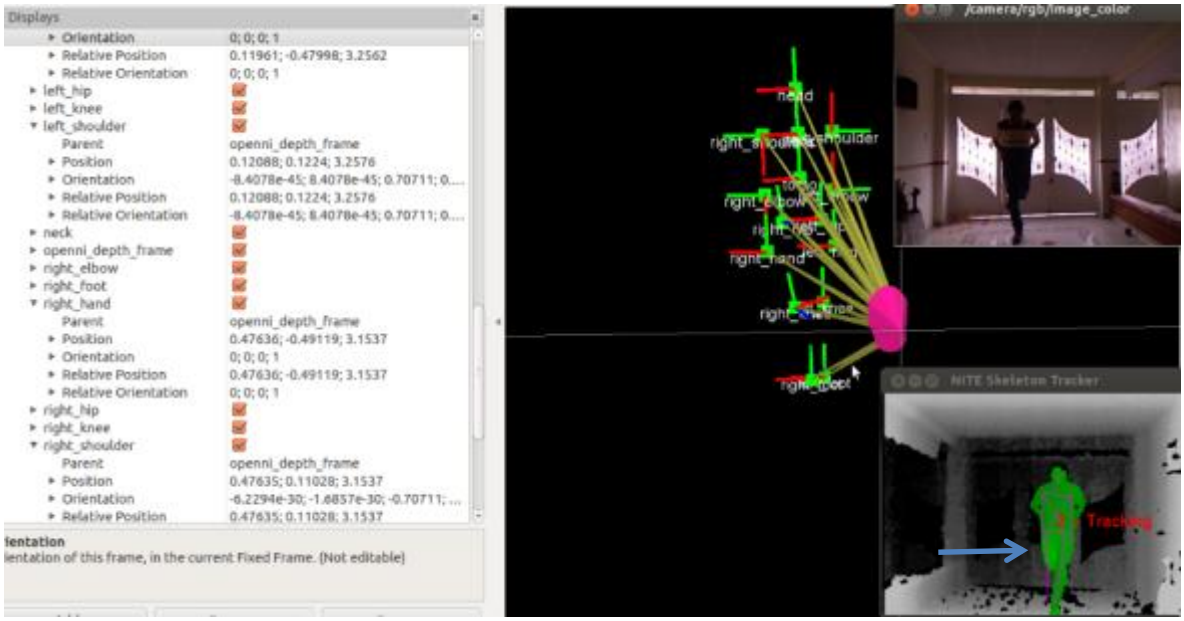


Figura 103. Posición incorrecta de los pies

En la figura 103 se muestra en la parte superior el cuerpo de un usuario con una pierna sobrepuesta sobre la otra utilizando la cámara RGB y en la parte inferior utilizando el OpenNI_Tracker se muestran las dos piernas hacia abajo, lo que indica que sus medidas están mal referenciadas y este valor no tiene la certeza esperada.

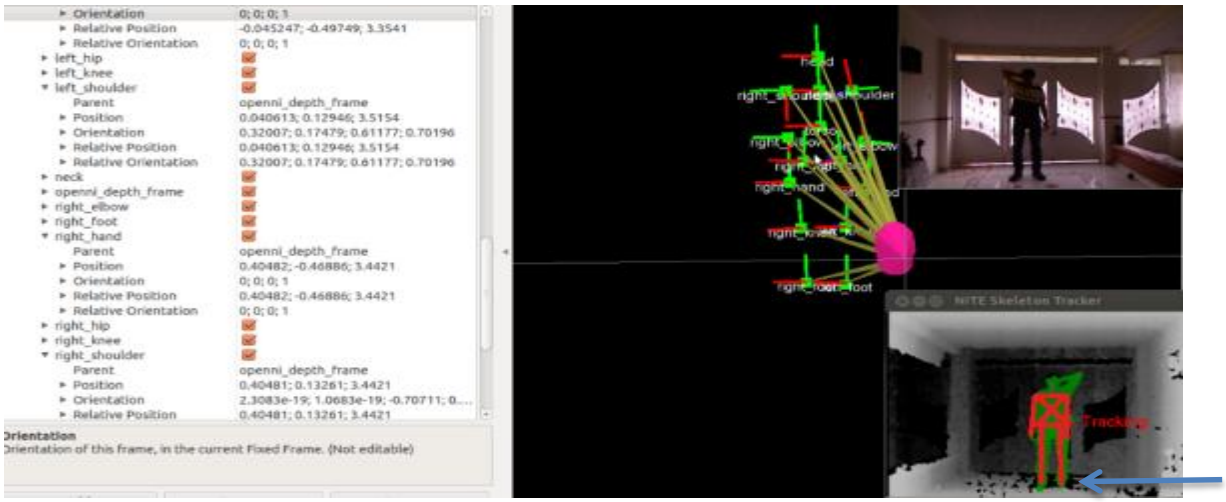


Figura 104. Posición incorrecta de la mano derecha.

En la figura 104, se visualiza que el usuario tiene en alto el brazo derecho y oculta su mano detrás de la cabeza, el OpenNI_Tracker referencia los dos brazos hacia abajo, lo que significa un error en la medida real, ya que estaría basada en la posición incorrecta de los brazos del usuario.

Figura 105. Posición incorrecta de las manos.

En la figura 105 se muestra un usuario con los dos brazos flexionados y al frente, el error presentado por el OpenNI_tracker es que se omite la parte del brazo desde el hombro hasta el codo y solo presenta la parte del codo hasta la mano.

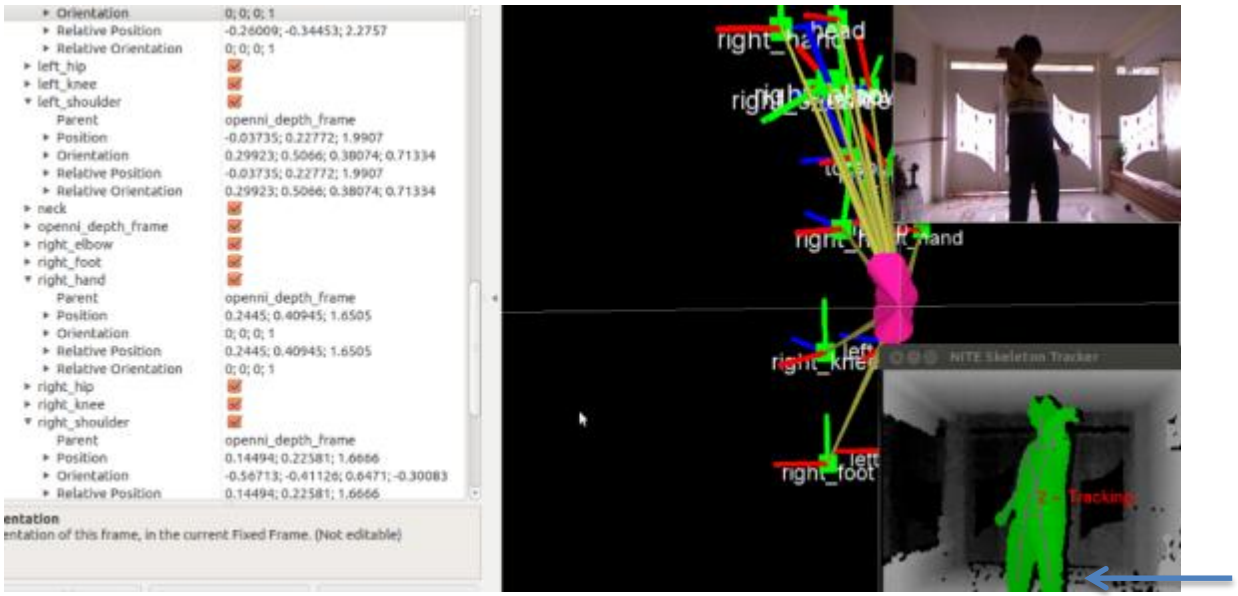


Figura 106. Error de posicionamiento frente a la cámara.

En la figura 106 se visualiza que el usuario está muy cerca de la cámara del Kinect y sus piernas son recortadas, lo que conlleva a un error de medida ya que este dato estaría fuera de la realidad de sus piernas.

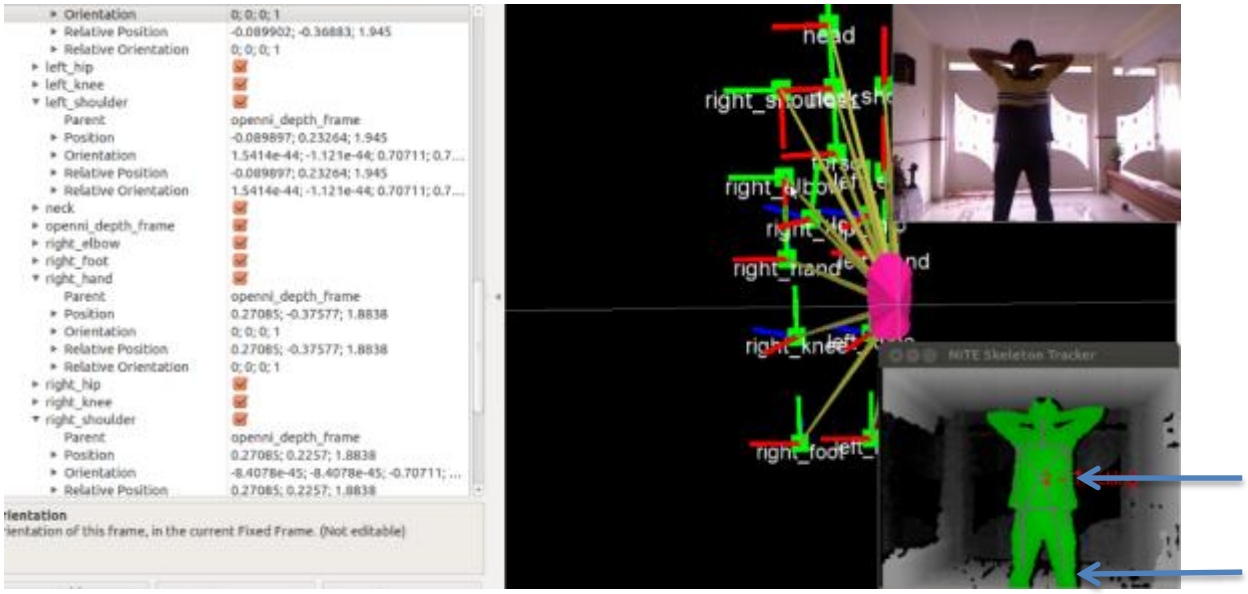


Figura 107. Presentación de errores múltiples (Manos y pies).

En la figura 107 se muestra que la persona tiene sus dos brazos detrás de la cabeza, esta posición presenta un doble error en la medida ya que OpenNI referencia sus dos brazos hacia abajo y sus piernas están recortadas.

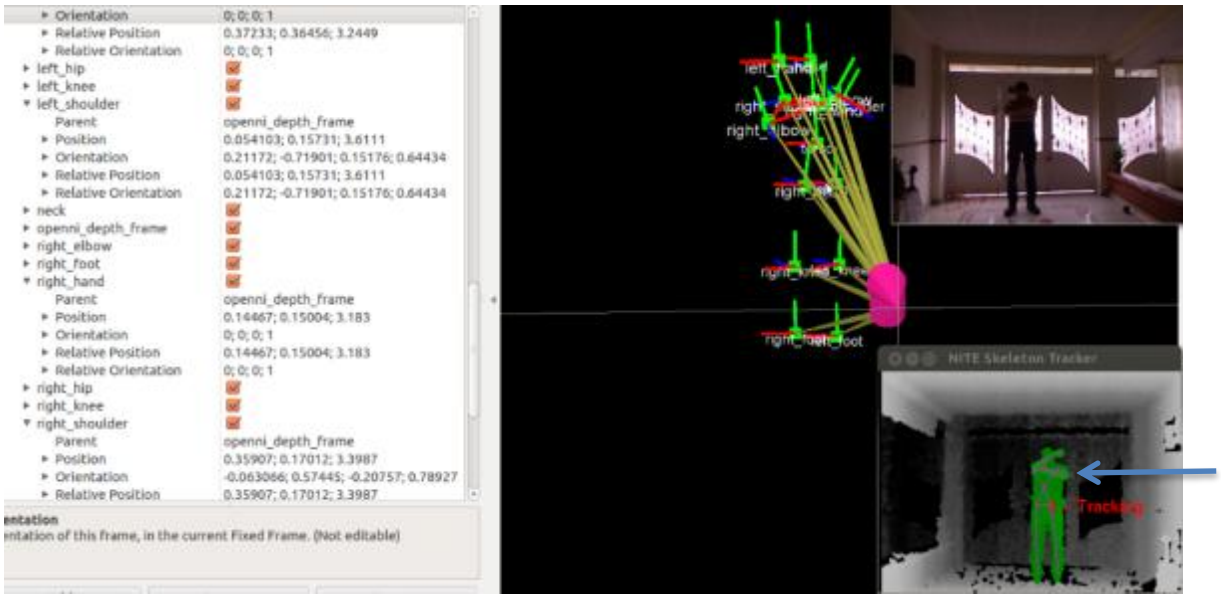


Figura 108. Presentación de errores múltiples (Manos y pies).

En la figura 108 se observa que el usuario está levantando un brazo a la altura de su cara y el otro cerca al hombro, se presenta un error en la medida de sus coordenadas ya que en el OpenNI_Tracker se sobreponen la articulación del hombro con la del brazo.

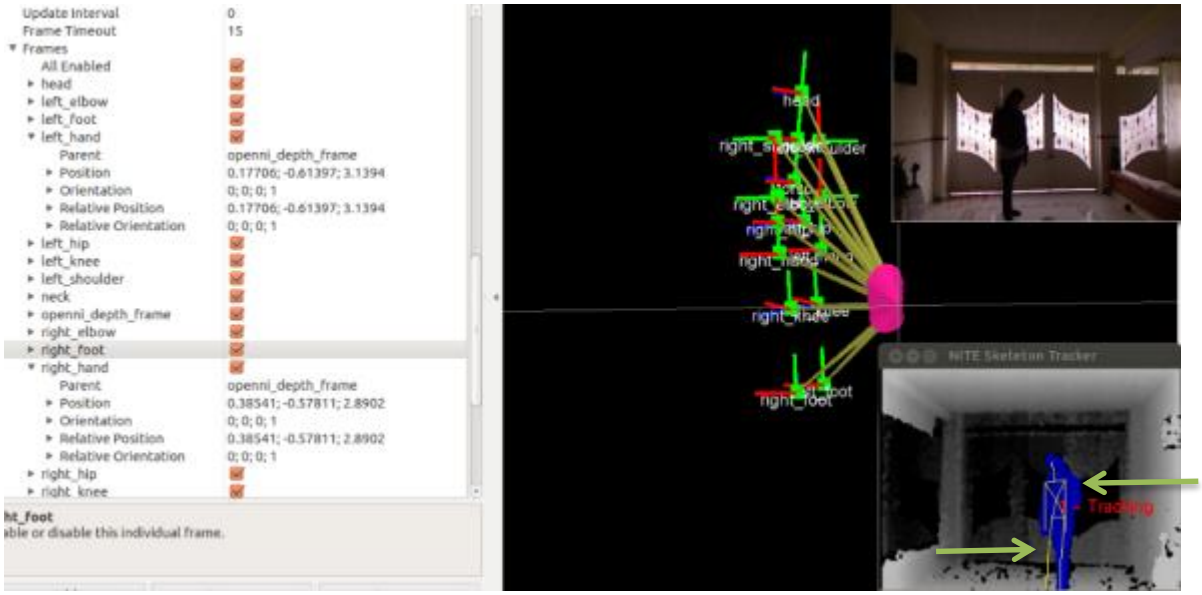


Figura 109. Presentación de errores múltiples (Manos y pies).

En la figura 109 se observa una presentación de errores múltiples en la mano y pie, el usuario tiene un pie delante del otro y uno de los brazos detrás de la espalda y el otro en posición derecha al costado del cuerpo, en el OpenNI_Tracker los dos brazos están en

línea recta así como las piernas como si la persona tuviera todas sus extremidades frente a la cámara.

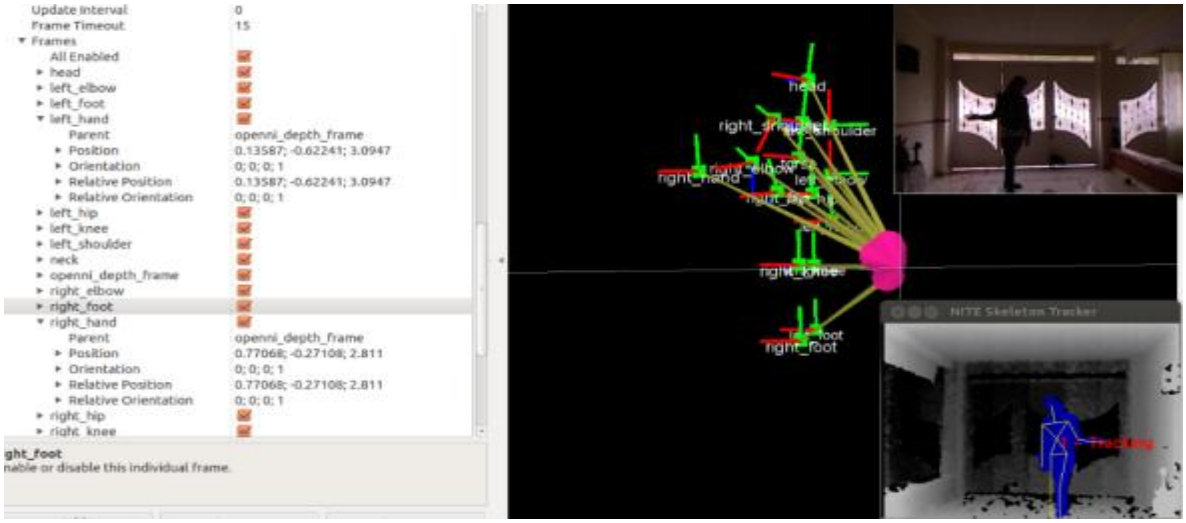


Figura 110. Corrección de la posición.

En la figura 110 se realiza una corrección de la posición de la figura 109, el brazo del usuario está al lado del cuerpo y sus piernas aunque estén muy cercanas están debidamente referenciadas.

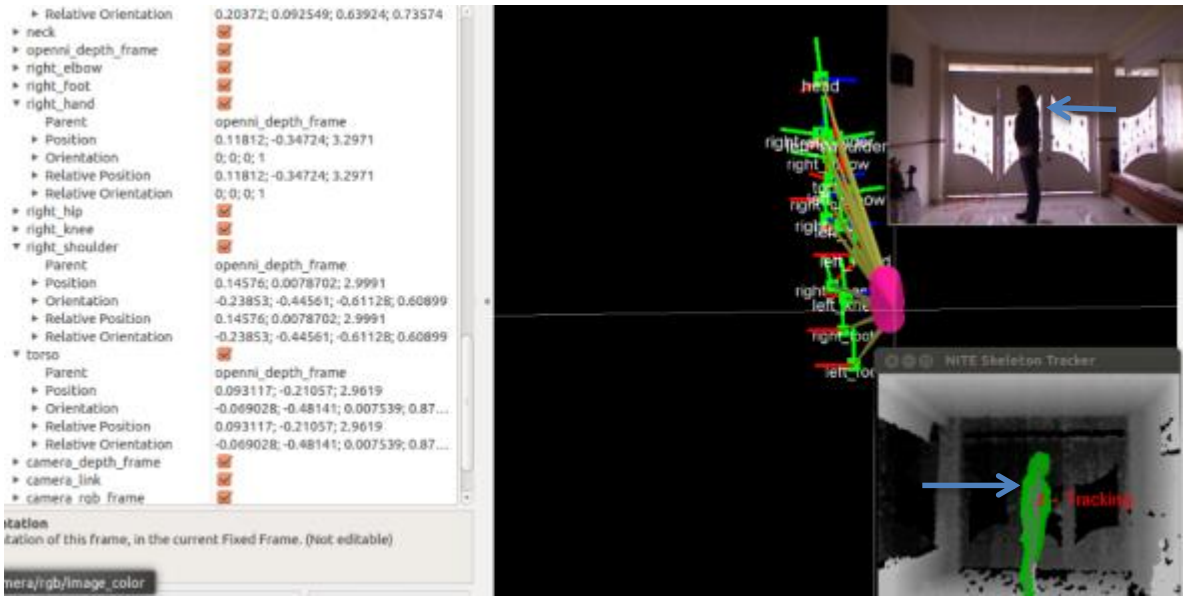


Figura 111. Posicionamiento erróneo para OpenNI_Tracker.

En la figura 111 se observa que la persona se encuentra de perfil frente a la cámara del Kinect, en esta posición desaparece un hombro y su respectivo brazo.

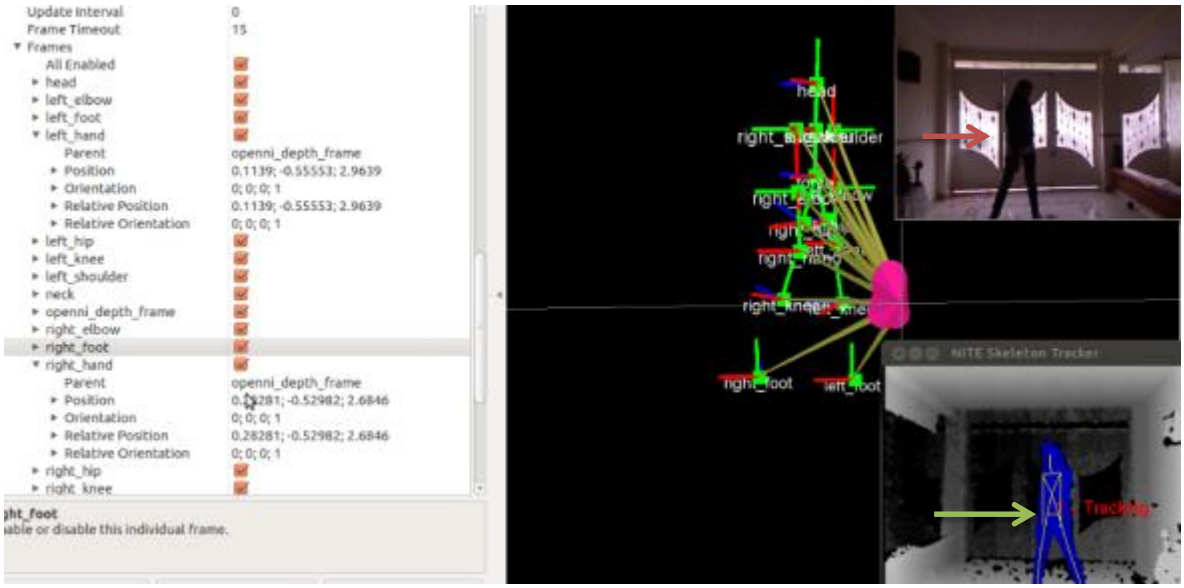


Figura 112. Posición de brazo oculto.

En la figura 112 se muestra un usuario con un brazo oculto y se observa en el OpenNI_Tracker que los dos brazos están en una posición derecha y a los lados del cuerpo lo que es falso.

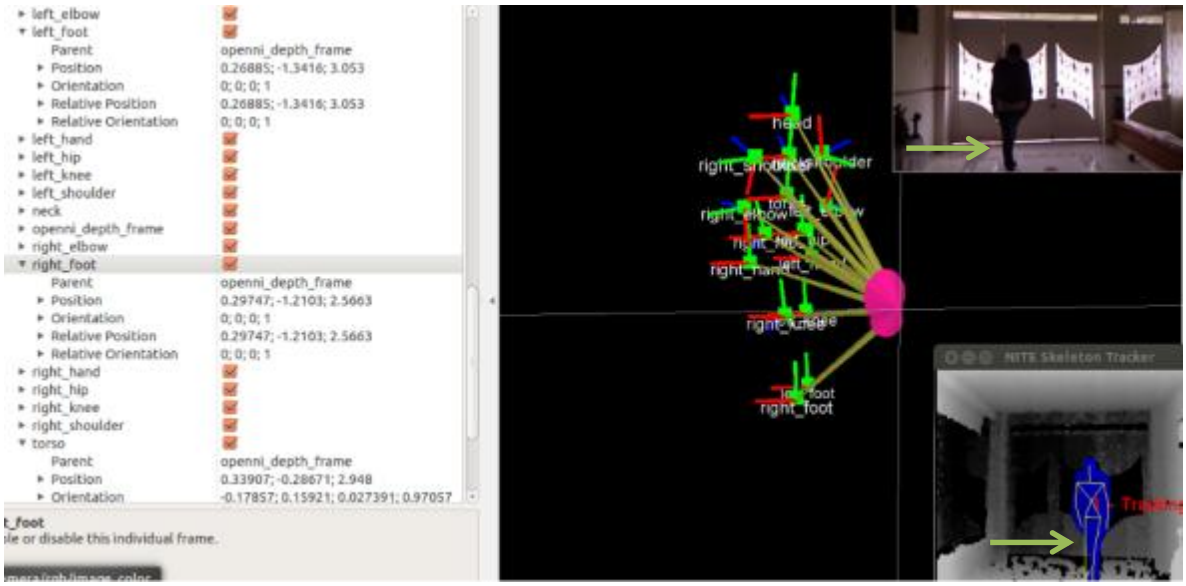


Figura 113. Posición de pie oculto.

En la figura 113 se puede observar que el usuario tiene un pie delante del otro, ocultando uno de ellos; en OpenNI_Tracker se muestran los dos pies separados y derechos.

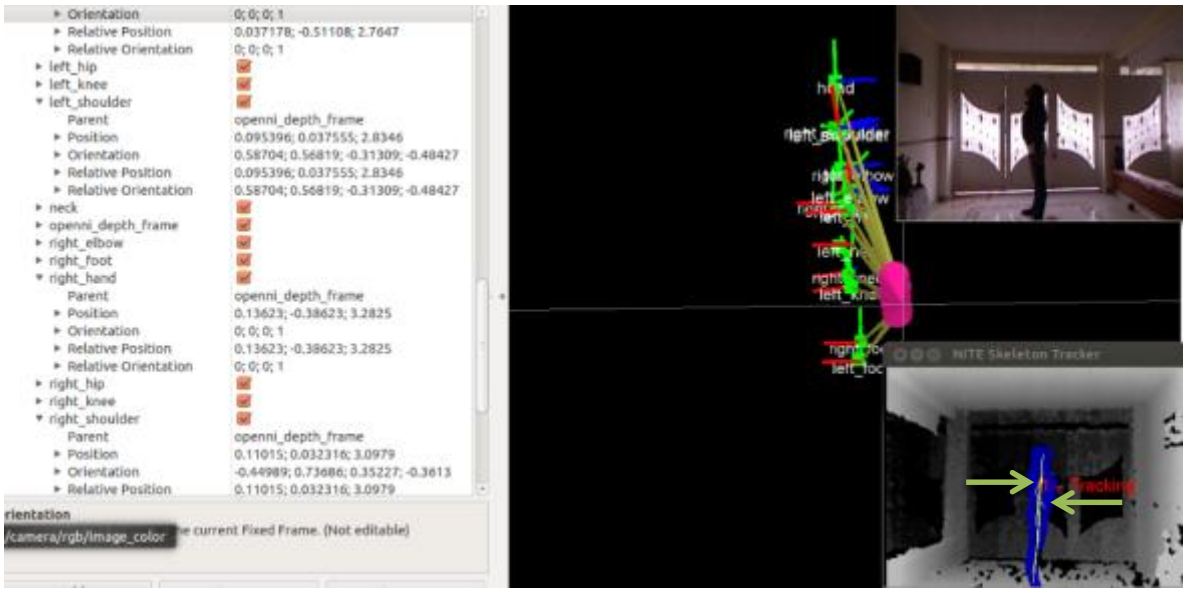


Figura 114. Prueba de torso y hombros.

En la figura 114 se visualiza que se pierden en su mayoría las articulaciones a medir como: los hombros, el torso, brazo y piernas, lo que hace que esta posición especifica se critica para efectos de medidas.

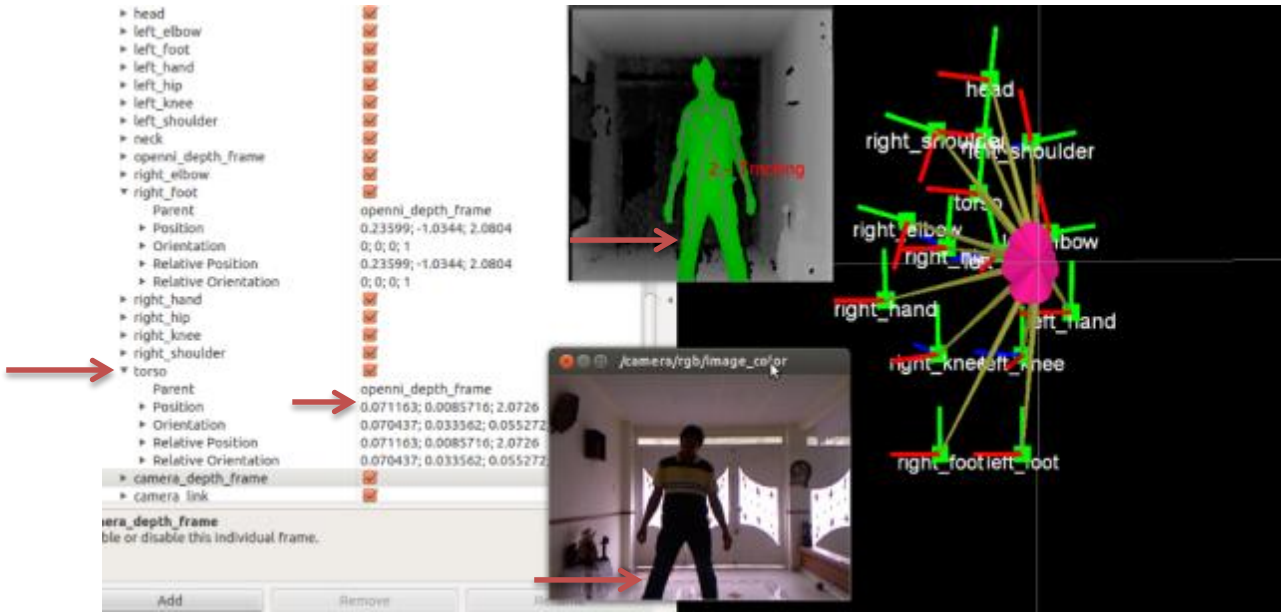


Figura 115. Referenciación de las piernas.

En la figura 115 el usuario está ubicado a 71 cm de la cámara del Kinect, sus piernas se ven por debajo de la rodilla utilizando la cámara RGB; OpenNI_Tracker muestra un error en la medida ya que solo se muestra una pequeña parte de ellas.

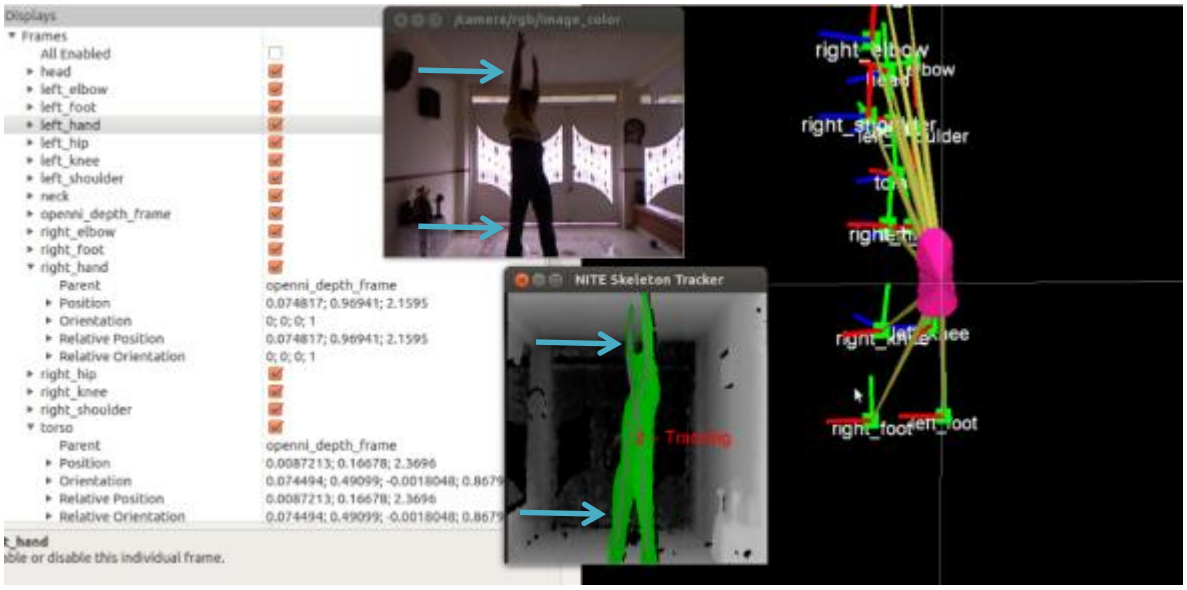


Figura 116. Brazos estables.

En la figura 116 se visualiza un usuario con las piernas separadas y sus dos brazos arriba de su cabeza, se presenta una medida estable en brazos y piernas, lo que indica que es una posición favorable que proporcionará valores certeros, debido a que muestran todas las articulaciones conforme a la cámara RGB.

Figura 117. Brazos inestables.

En la figura 117 muestra una persona muy cerca de la cámara Kinect a 51 cm de ella, con sus piernas separadas y sus brazos levantados; uno de ellos registrado en su totalidad y en el otro recortada su mano; en el OpenNI_Tracker se ve el error al tener el brazo recortado, hacia abajo, derecho y a un costado del cuerpo.

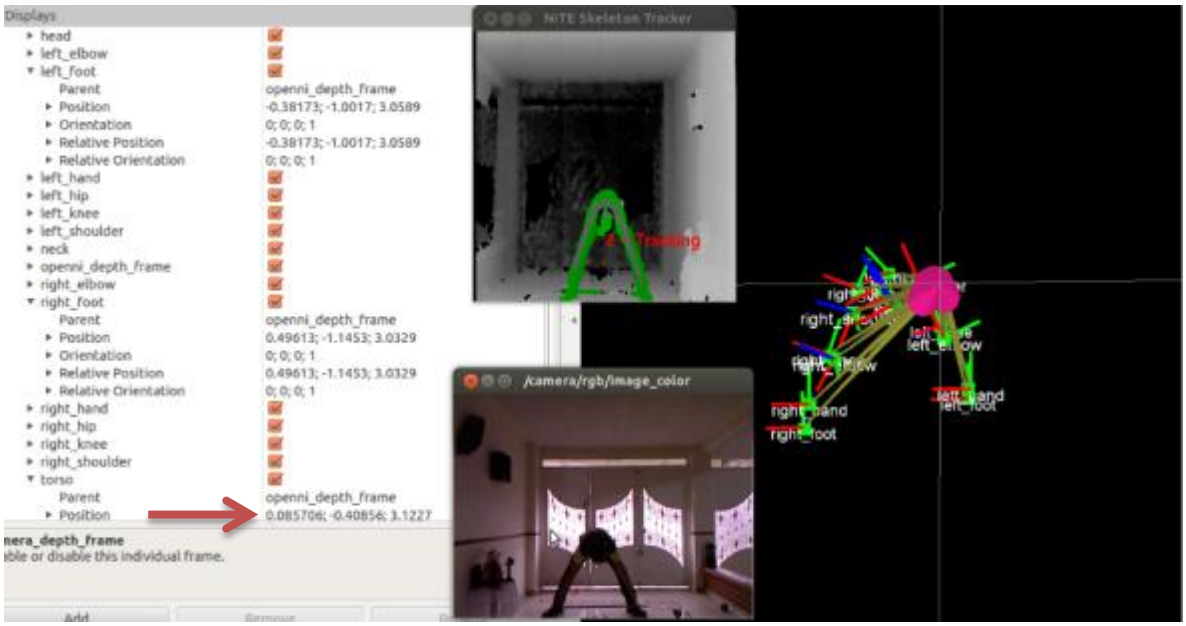


Figura 118. Pies estables.

En la figura 118 se registra una persona con una separación de pies, la medida en OpenNI_Tracker en el eje x es de 85 cm y utilizando un flexómetro es de 87 cm; valores que se asemejan de software e instrumento, llevando a una magnitud estable en pies.

Figura 119. Calibración del punto cero.

En la figura 119 se observa la calibración del punto cero en un registro de usuario, se muestra el torso ubicado en la posición cero del eje x; esta calibración se deberá realizar al iniciar el proceso para establecer desde que distancia es recomendable tomar las medidas.

Figura 120. Posición manos inestables.

En la figura 120 se registra un usuario con los brazos levantados y sus manos unidas, para OpenNI_Tracker esta posición es incorrecta y como efecto de esto, muestra los dos brazos hacia abajo.

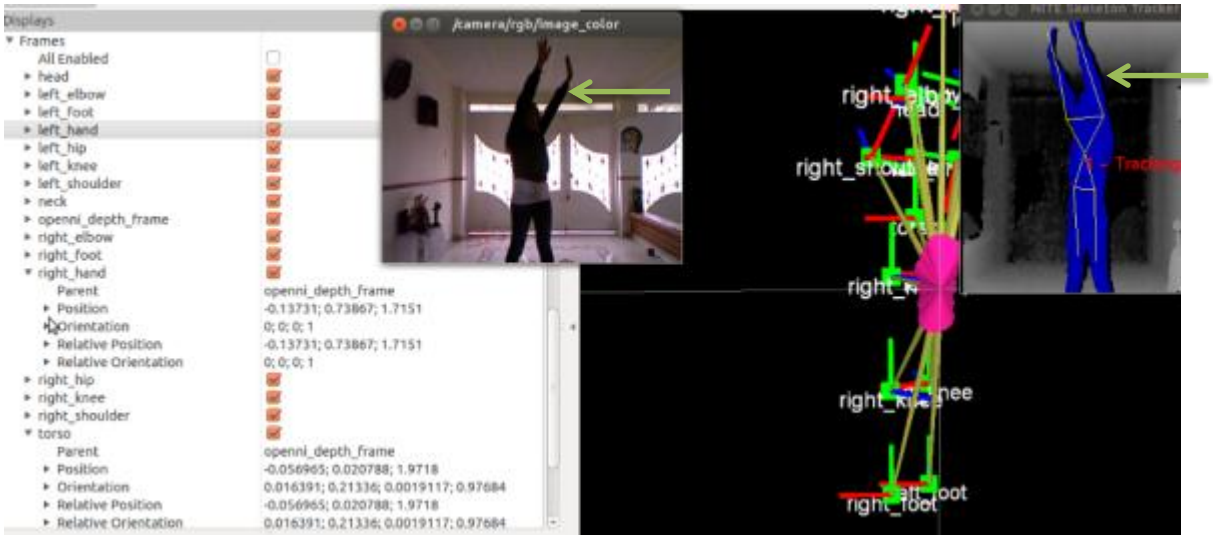


Figura 121. Posición manos estables.

En la figura 121 se observa una persona con los brazos levantados y sus manos separadas, obteniendo el mismo registro con el OpenNI_Tracker, lo que conlleva a establecer que es una posición de manos estable para este paquete.

Figura 122. Posición manos inestable.

En la figura 122 se registra una persona con los brazos levantados con cierta inclinación, sus manos están muy cercanas; esta posición genera un error en el paquete OpenNI_Tracker de ROS, ya que se visualizan los brazos contrarios a la imagen real.

5.2 PAQUETE TF BROADCASTER Y LISTENER.

5.2.1 TF Broadcaster. Envía las coordenadas de frame al resto del sistema. Se pueden construir varios broadcaster y enfocarse en diferentes partes del robot. Para este caso se desarrolló uno solo utilizando el OpenNI, como se muestra a continuación.

```cpp
// openni_tracker.cpp

#include <ros/ros.h>
#include <ros/package.h>
#include <tf/transform_broadcaster.h>
#include <kdl/frames.hpp>

#include <XnOpenNI.h>
#include <XnCodecIDs.h>
#include <XnCppWrapper.h>

using std::string;

xn::Context        g_Context;
xn::DepthGenerator g_DepthGenerator;
xn::UserGenerator  g_UserGenerator;

XnBool g_bNeedPose   = FALSE;
XnChar g_strPose[20] = "";

void XN_CALLBACK_TYPE User_NewUser(xn::UserGenerator& generator, XnUserID nId, void* pCookie) {
        ROS_INFO("New User %d", nId);

        if (g_bNeedPose)
                g_UserGenerator.GetPoseDetectionCap().StartPoseDetection(g_strPose, nId);
        else
                g_UserGenerator.GetSkeletonCap().RequestCalibration(nId, TRUE);
}

void XN_CALLBACK_TYPE User_LostUser(xn::UserGenerator& generator, XnUserID nId, void* pCookie) {
        ROS_INFO("Lost user %d", nId);
}

void XN_CALLBACK_TYPE UserCalibration_CalibrationStart(xn::SkeletonCapability& capability, XnUserID nId, void* pCookie) {
        ROS_INFO("Calibration started for user %d", nId);
}
```

Generación de datos y mapa de profundidad

Detección de pose

```cpp
XnSkeletonJointOrientation joint_orientation;
g_UserGenerator.GetSkeletonCap().GetSkeletonJointOrientation(user, joint, joint_orientation);

XnFloat* m = joint_orientation.orientation.elements;
KDL::Rotation rotation(m[0], m[1], m[2],
                       m[3], m[4], m[5],
                       m[6], m[7], m[8]);
double qx, qy, qz, qw;
rotation.GetQuaternion(qx, qy, qz, qw);

char child_frame_no[128];
snprintf(child_frame_no, sizeof(child_frame_no), "%s_%d", child_frame_id.c_str(), user);

tf::Transform transform;
transform.setOrigin(tf::Vector3(x, y, z));
transform.setRotation(tf::Quaternion(qx, -qy, -qz, qw));

// #4994
tf::Transform change_frame;
change_frame.setOrigin(tf::Vector3(0, 0, 0));
tf::Quaternion frame_rotation;
frame_rotation.setEulerZYX(1.5708, 0, 1.5708);
change_frame.setRotation(frame_rotation);

transform = change_frame * transform;

br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), frame_id, child_frame_no));
```

Adquisición de Joint

Rotación

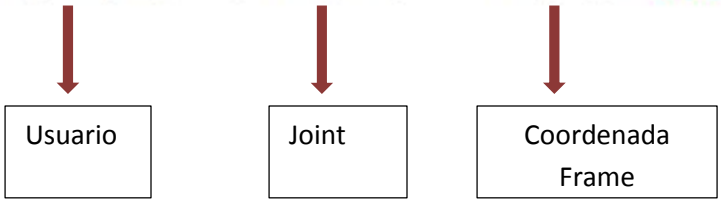
Vectores de posición X,Y,Z y Rotacional qx,qy,qz

Función TF Transform Frame

```cpp
d publishTransforms(const std::string& frame_id) {
  XnUserID users[15];
  XnUInt16 users_count = 15;
  g_UserGenerator.GetUsers(users, users_count);

  for (int i = 0; i < users_count; ++i) {
      XnUserID user = users[i];
```

Publish Transforms e ID

```
for (int i = 0; i < users_count; ++i) {
    XnUserID user = users[i];
    if (!g_UserGenerator.GetSkeletonCap().IsTracking(user))
        continue;


    publishTransform(user, XN_SKEL_HEAD,           frame_id, "head");
    publishTransform(user, XN_SKEL_NECK,           frame_id, "neck");
    publishTransform(user, XN_SKEL_TORSO,          frame_id, "torso");

    publishTransform(user, XN_SKEL_LEFT_SHOULDER,  frame_id, "left_shoulder");
    publishTransform(user, XN_SKEL_LEFT_ELBOW,     frame_id, "left_elbow");
    publishTransform(user, XN_SKEL_LEFT_HAND,      frame_id, "left_hand");

    publishTransform(user, XN_SKEL_RIGHT_SHOULDER, frame_id, "right_shoulder");
    publishTransform(user, XN_SKEL_RIGHT_ELBOW,    frame_id, "right_elbow");
    publishTransform(user, XN_SKEL_RIGHT_HAND,     frame_id, "right_hand");

    publishTransform(user, XN_SKEL_LEFT_HIP,       frame_id, "left_hip");
    publishTransform(user, XN_SKEL_LEFT_KNEE,      frame_id, "left_knee");
    publishTransform(user, XN_SKEL_LEFT_FOOT,      frame_id, "left_foot");

    publishTransform(user, XN_SKEL_RIGHT_HIP,      frame_id, "right_hip");
    publishTransform(user, XN_SKEL_RIGHT_KNEE,     frame_id, "right_knee");
    publishTransform(user, XN_SKEL_RIGHT_FOOT,     frame_id, "right_foot");
}
```




```
if (g_UserGenerator.GetSkeletonCap().NeedPoseForCalibration()) {
        g_bNeedPose = TRUE;
        if (!g_UserGenerator.IsCapabilitySupported(XN_CAPABILITY_POSE_DETECTION)) {
                ROS_INFO("Pose required, but not supported");
                return 1;
        }

        XnCallbackHandle hPoseCallbacks;
        g_UserGenerator.GetPoseDetectionCap().RegisterToPoseCallbacks(UserPose_PoseDetected, NULL, NULL, hPoseCallbacks);

        g_UserGenerator.GetSkeletonCap().GetCalibrationPose(g_strPose);
}

g_UserGenerator.GetSkeletonCap().SetSkeletonProfile(XN_SKEL_PROFILE_ALL);

nRetVal = g_Context.StartGeneratingAll();
CHECK_RC(nRetVal, "StartGenerating");

ros::Rate r(30);


ros::NodeHandle pnh("~");
string frame_id("openni_depth_frame");
pnh.getParam("camera_frame_id", frame_id);

while (ros::ok()) {
        g_Context.WaitAndUpdateAll();
        publishTransforms(frame_id);
        r.sleep();
}

g_Context.Shutdown();
return 0;
```

Envió del Joint

5.2.2 TF Listener. Es un paquete complementario a TF Broadcaster, el cual cumple la función de recibir y almacenar todas las coordenadas de frame que se transmiten en el sistema, y realiza consultas para transformadas especificas entre frame; para este proyecto se utilizará el código 3D_Skeleton_Master con el siguiente contenido.

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <visualization_msgs/Marker.h>

#include <cmath>
////// MACROS
#define ODF 0    // openni_depth_frame
#define HEAD 1
#define NECK 2
#define TORSO 3
#define LEFT_SHOULDER 4
#define LEFT_ELBOW 5
#define LEFT_HAND 6
#define RIGHT_SHOULDER 7
#define RIGHT_ELBOW 8
#define RIGHT_HAND 9
#define LEFT_HIP 10
#define LEFT_KNEE 11
#define LEFT_FOOT 12
#define RIGHT_HIP 13
#define RIGHT_KNEE 14
#define RIGHT_FOOT 15

int sequence[34]={HEAD,NECK,
LEFT_SHOULDER,LEFT_ELBOW,
LEFT_ELBOW,LEFT_HAND,
RIGHT_SHOULDER,RIGHT_ELBOW,
RIGHT_ELBOW,RIGHT_HAND,
LEFT_HIP,LEFT_KNEE,
LEFT_KNEE,LEFT_FOOT,
RIGHT_HIP,RIGHT_KNEE,
RIGHT_KNEE,RIGHT_FOOT,
LEFT_SHOULDER,TORSO,
RIGHT_SHOULDER,TORSO,
LEFT_SHOULDER,LEFT_HIP,
RIGHT_SHOULDER,RIGHT_HIP,
LEFT_SHOULDER,RIGHT_SHOULDER,
RIGHT_HIP,LEFT_HIP,
TORSO,RIGHT_HIP,
TORSO,LEFT_HIP};
```

Librerías: general (ros.h), tf listener y visualización del mensaje.

Directivas, identificadores de cadenas en C++.

Define los 15 puntos articulados del cuerpo humano.

Crea las líneas entre puntos articulados formando el esqueleto.

```
float x_value[15];
float y_value[15];
float z_value[15];
//// stores the name of tf frames
char array[16][20]={"/openni_depth_frame","/head_1","/neck_1","/torso_1","/left_shoulder_1","/left_elbow_1","/left_hand_1","/right
int main( int argc, char** argv )
{ int i=0;
  ros::init(argc, argv, "skeleton");
  ros::NodeHandle n,node;
  tf::TransformListener listener;
  listener.waitForTransform("/openni_depth_frame", "/head_1", ros::Time(), ros::Duration(5.0));

ros::Publisher marker_pub = n.advertise<visualization_msgs::Marker>("visualization_marker", 10);

  ros::Rate r(30);
  while (node.ok())
  {
```

Creación de array para los puntos coordenados x,y,z.

Transform Listener

```
   tf::StampedTransform transform;
//// this loop calculates all the points and store it
        for(i=0;i<=14;i++)
                { try{
                      listener.lookupTransform(array[0], array[i+1],
                                  ros::Time(0), transform);
                    }
                 catch (tf::TransformException ex)
                {  ROS_ERROR("%s",ex.what());}
                x_value[i]=transform.getOrigin().x();
                y_value[i]=-transform.getOrigin().y();
                z_value[i]=transform.getOrigin().z();
                //ROS_INFO(" %f %f %f ",x_value[i],z_value[i],z_value[i]);
                }
```

Publicación de Transformada

Evalúa el punto, lo guarda y publica

```
visualization_msgs::Marker points,line_list;
points.header.frame_id = line_list.header.frame_id = array[0];
points.header.stamp = line_list.header.stamp = ros::Time::now();
points.ns = line_list.ns = "points_and_lines";
points.action =line_list.action = visualization_msgs::Marker::ADD;
points.pose.orientation.w = line_list.pose.orientation.w = 1.0;
points.id = 0;
line_list.id = 1;
points.type = visualization_msgs::Marker::POINTS;
line_list.type = visualization_msgs::Marker::LINE_LIST;
```

Visualización del mensaje.

Visualización del esqueleto (líneas y puntos)

```
    points.scale.x = 0.04;
    points.scale.y = 0.04;

line_list.scale.x = 0.01;
    // Points are green
    points.color.g = 1.0f;
    points.color.a = 1.0;
    // Line list is red
    line_list.color.r = 1.0f;
    line_list.color.a = 1.0;
    // Create the vertices for the points and lines
geometry_msgs::Point p;

//// this for loop displays points on joints cordinates
        for (i=0; i <=14; i++)
        {
        p.x = x_value[i];
        p.y = y_value[i];
        p.z = z_value[i];
        points.points.push_back(p);
        }
//// this for loop makes lines between two points
        for(i=0;i<34;i++)
        {
        p.x = x_value[(sequence[i]-1)];
        p.y = y_value[(sequence[i]-1)];
        p.z = z_value[(sequence[i]-1)];
        line_list.points.push_back(p);
        }

   marker_pub.publish(points);
   marker_pub.publish(line_list);

    r.sleep();


  }
}
```

Definición de líneas, puntos y colores para formar el esqueleto

Puntos coordenados

Líneas entre dos puntos

Publicación de puntos y líneas (Esqueleto)

Este paquete cumple la función de recibir los 100 joints identificados según el usuario, con el objetivo de ser utilizados posteriormente por un robot dependiendo de los requerimientos.

5.3 PROMEDIO DE DATOS. Debido a la magnitud de datos que se pueden generar, se requiere realizar un promedio de las magnitudes obtenidas, para realizar un buen manejo de los joints obtenidos.

Se utilizó el siguiente código para realizar el promedio:

Guardar los valores de las transformadas en arrays, x, y, z con 100 valores, que representan los joints transformados:

```
float x_value[15];
float y_value[15];
float z_value[15];
```

Guardar los nombres de los tf, así:

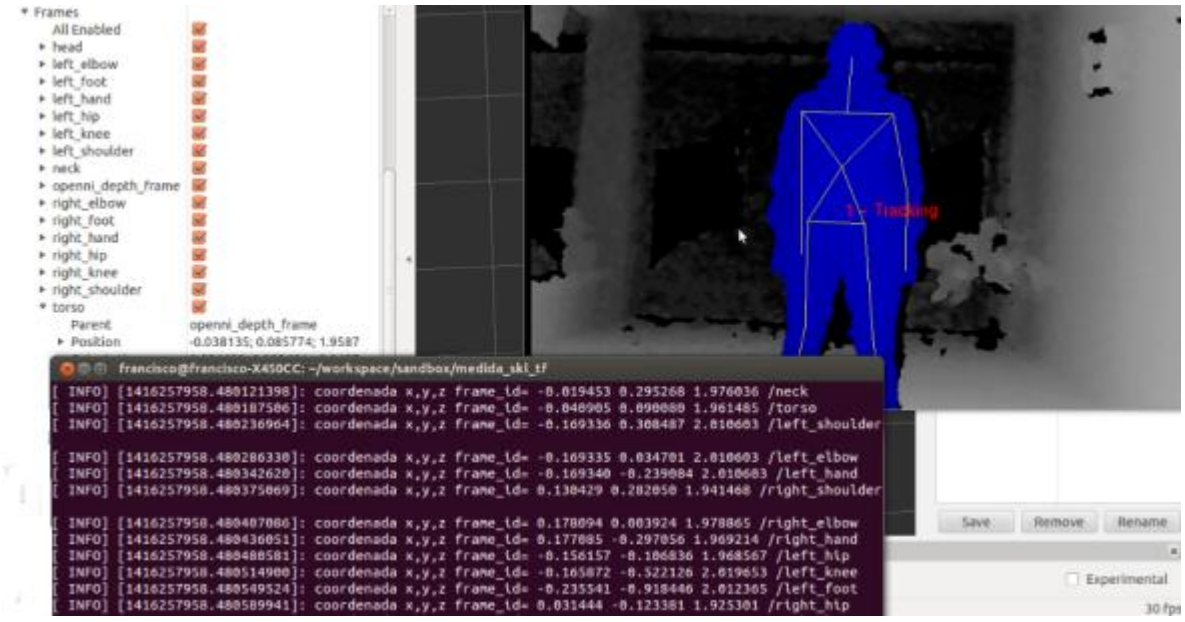


Figura 123. Pantallazo de coordenadas.

5.4 TOMA DE MEDIDAS. Para la toma de medidas de los joints, se utilizó un nivel láser en cruz que simula los dos ejes coordenados X y Y; y un medidor de distancia láser proyectado al usuario para determinar Z.

Lista de Materiales

- **Medidor de distancias por laser**

Figura 124. Medidor de distancia laser [102].

Datos Técnicos

- Método de medición: Tecnología LÁSER
- Rango de medición: 0,2 - 20 m
- Precisión de las mediciones (típica) : ± 3,0 mm
- Duración de la medición (típica): < 0,5 s
- Unidad de medición más baja : 1 mm
- Clase de láser: 2
- Tipo de láser: 650 nm
- Potencia de salida máxima: < 1 mW
- Apagado: automático después de ± 5 min
- Suministro de potencia: 4 baterías AAA (LR03) de 1,5 V
- Peso: 0,18 kg

- **Dispositivo de Nivelación laser**

Figura 125. Dispositivo de nivel laser [103].

Datos Técnicos

- Tipo de láser: 635 nm
- Clase de láser: 2
- Potencia de salida máxima: < 1 mW
- Nivelación automática: sí
- Precisión a 10 metros: 5 mm
- Alcance: 10 m
- Suministro de potencia: Baterías/AA
- Ángulo de compensación de autonivelación: +/- 4 °
- Tiempo de nivelación: 6 s
- Suministro de potencia : 4 baterías AA (LR6) de 1,5 V
- Temperatura de funcionamiento: -1 °C -40 °C
- Peso: 0,47 kg
- 1 línea láser en cruz (1 línea horizontal y una línea vertical) y 1 línea vertical en ángulo de 90°; las líneas pueden proyectarse por separado
- Adaptador roscado para trípode de 1/4"
- Función de pendiente: sí

- **Maniquí**
- **Flexometro**
- **Computador Hp, Asus**
- **Kinect**

El punto cero-cero se establecerán coincidiendo el punto del nivel láser con el proyectado con la cámara Kinect, como se indica en la figura 126.

Figura 126. Punto cero entre dispositivos.

Esta calibración se realiza de manera frontal, coincidiendo el punto cero-cero emitido por el nivel láser con la cámara Kinect, como se observa en la figura 127.

Figura 127. Visual de punto cero.

El nivel láser también emite un rayo en sus paredes laterales las cuales son de utilidad para ser alineado con el Kinect y proporcionar una medida lo más exacta y precisa, como se muestra en la figura 128

Figura 128. Rayo lateral en relación con Kinect.

Una vez realizada esta calibración manual, se procederá a emitir el rayo sobre una superficie visible y para esto se utilizará otro elemento importante como es las gafas infrarrojas, como se indica en la figura 129

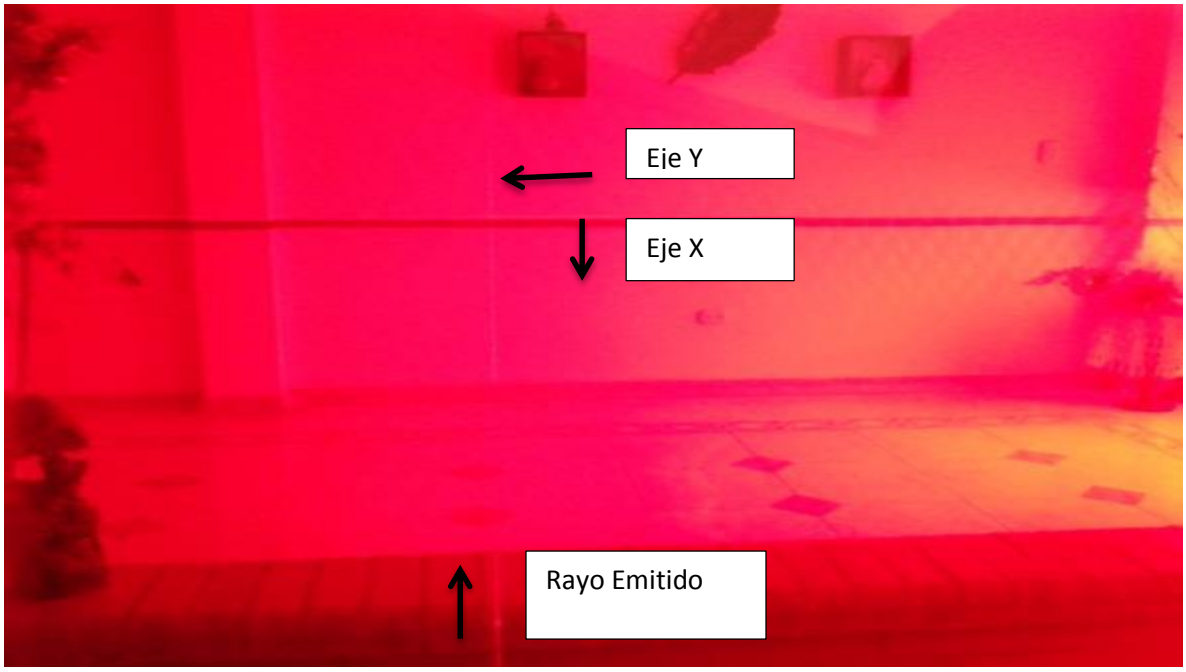


Figura 129. Identificación de rayo emitido y ejes X y Y.

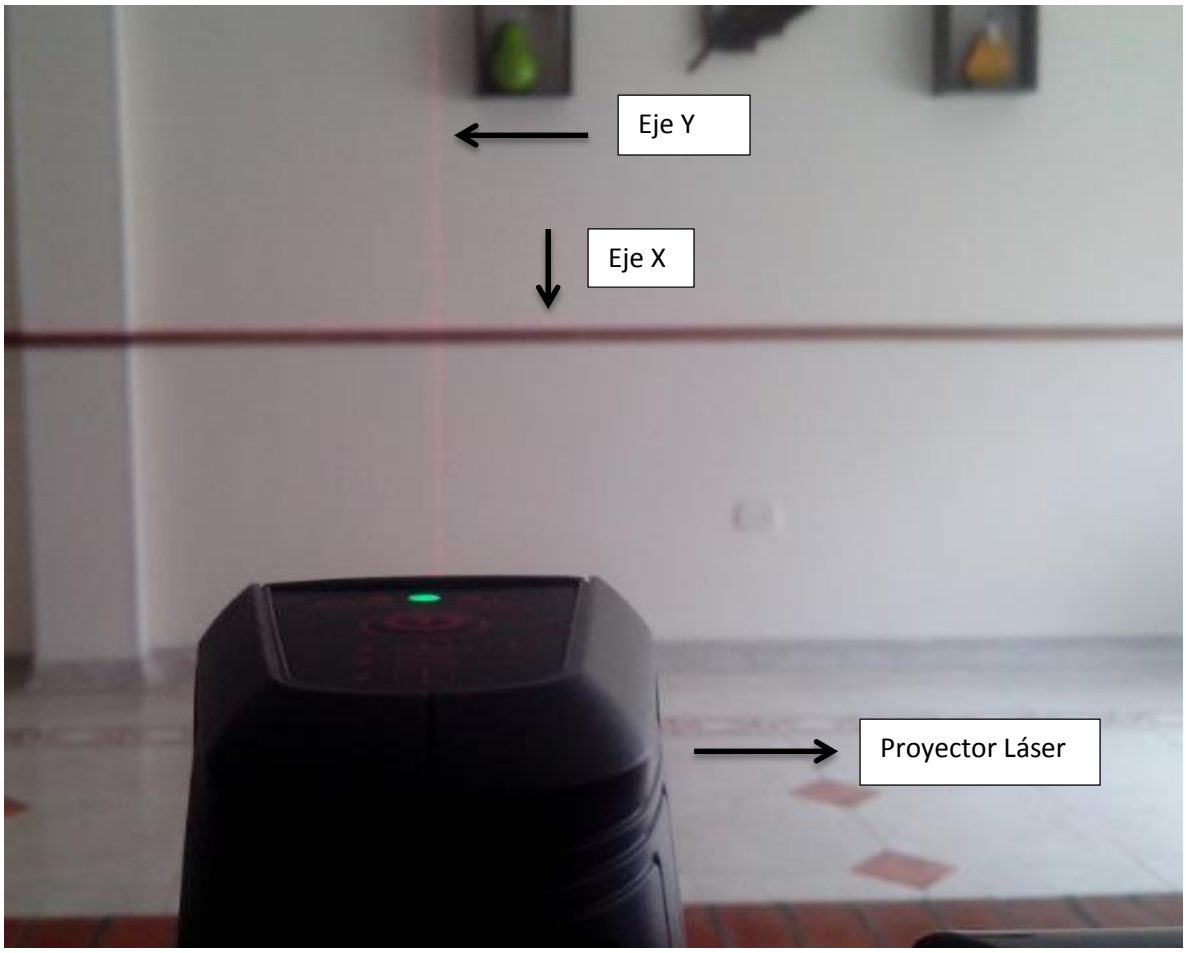


Figura 130. Proyector laser y proyección de ejes X y Y.

Ahora se proyectará el láser sobre un usuario para tener la certeza de calibración del punto cero-cero emitido sea real, que coincida el emitido por el Kinect y el nivel, así como se indica en la figura 131.

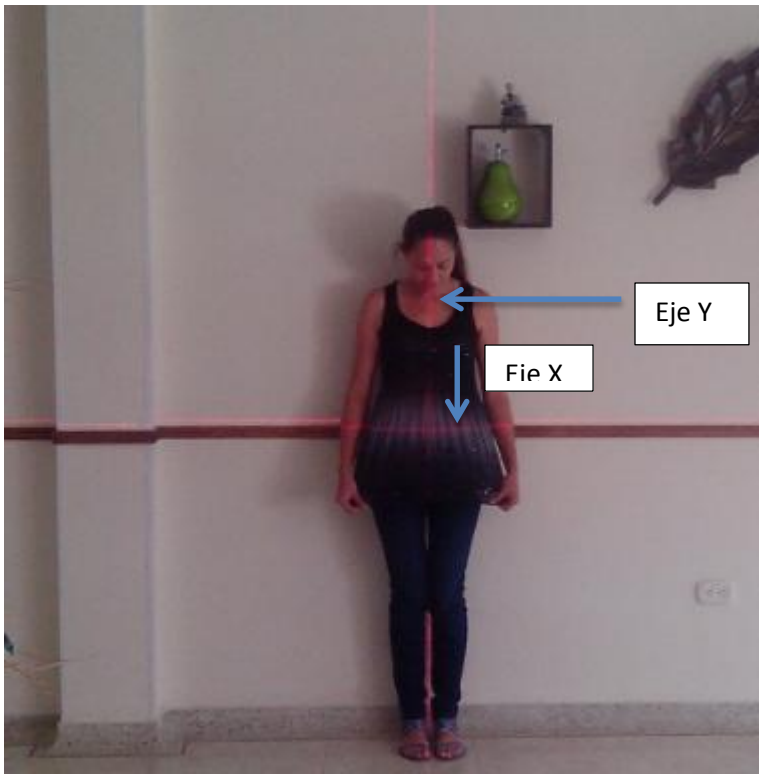


Figura 131. Identificación en usuario de ejes coordenados.

6. RESULTADOS

Se estableció por código la toma de 100 muestras de cada joint y 10 registros reales del punto sensado; información que se guardó para realizar el tratamiento de error RMS tanto de puntos individuales como general, lo cual se consiguió utilizando un maniquí que simulo la forma de un cuerpo femenino, como se indica en la figuras siguientes.

Figura 132. Visual del esqueleto, datos y promedio.

Para realizar esta visualización de los 100 puntos en las coordenadas X , Y y Z, así como el promedio de datos y la identificación del Joint, se construyó un paquete en ROS utilizando tf Listener que permite tener acceso a las transformaciones de frame (tf) para ser transmitidas al paquete de destino; así mismo también se utilizó la visualización del esqueleto utilizando el controlador Nite para asegurar una buena medida garantizando la formación del esqueleto en su totalidad.

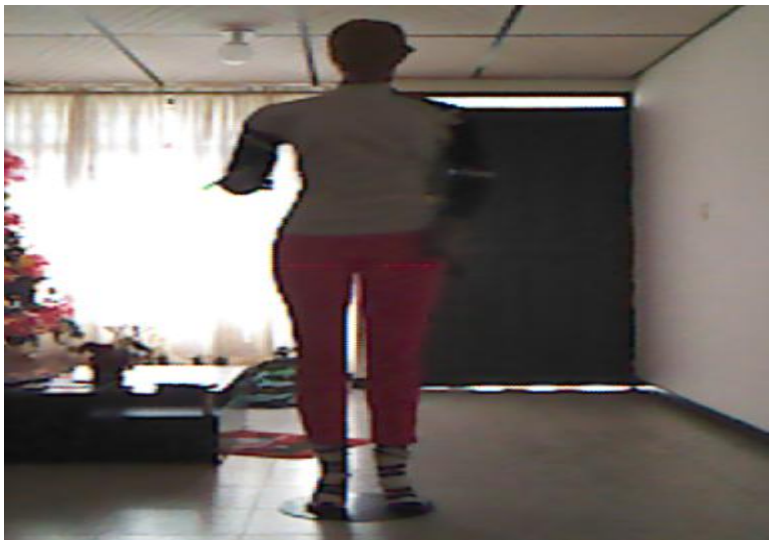


Figura 133. Maniquí usado para toma de datos.

Debido a la dificultad de mantener en reposo y fijos algunas partes del cuerpo humano se recurrió a un maniquí para tener la certeza de una medida confiable.

Una vez obtenido las quince articulaciones, se procedió a realizar el error RMS utilizando la siguiente formula.

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

Ecuación 14. Error RMS.

Se agruparon todos los joints en una tabla y se calculó el vector promedio medida real:

JOINTS	VECTOR PROMEDIO MEDIDA CON LASER		
	X	Y	Z
CABEZA	-0,0261	0,7586	2,5977
CUELLO	-0,1094	0,5558	2,7536
HOMBRO DERECHO	0,0398	0,0791	2,1879
HOMBRO IZQUIERDO	-0,1708	0,4897	1,9564
CODO DERECHO	0,2459	-0,1117	1,9197
CODO IZQUIERDO	-0,0377	0,2902	2,2784
TORSO	-0,2445	0,2498	2,0920
CADERA DERECHA	-0,0114	0,1478	2,2236
CADERA IZQUIERDA	-0,0006	0,2405	1,3844
MANO DERECHA	0,1779	0,4969	2,2746
MANO IZQUIERDA	-0,2116	0,0337	2,6121
RODILLA DERECHA	0,0371	-0,3750	2,3321
RODILLA IZQUIERDA	-0,1282	-0,3715	2,1272
PIE DERECHO	-0,0860	-0,7950	3,1047
PIE IZQUIERDA	-0,1398	-0,8203	2,8221

Tabla 3. Vector promedio medida con láser.

También se realizó un vector promedio de la medida del Kinect a todos los puntos:

VECTOR PROMEDIO MEDIDA KINECT		
X	Y	Z
-0,0241	0,7567	2,5860
-0,1010	0,5415	2,7419
0,0333	0,0631	2,1612
-0,1878	0,4790	1,9535
0,2444	-0,1529	1,8861
-0,0399	0,2893	2,2948
-0,2409	0,2475	2,0948
-0,0263	0,1288	2,2251
-0,0047	0,2398	1,3948
0,1716	0,4969	2,2617
-0,2211	0,0383	2,6222

0,0382	-0,3971	2,3198
-0,1163	-0,3935	2,1303
-0,0783	-0,7949	3,0136
-0,1510	-0,8190	2,8439

Tabla 4. Vector promedio medida Kinect.

Debido al tratamiento de los datos y a la utilización de instrumentos, se presentan una serie de errores que son el reflejo de la dificultad que se presentó al medir determinados joints, la alineación entre el Kinect y el Laser, los movimientos presentados repercuten en la deformación del esqueleto entre otros.

Es así como se relacionan algunos errores:

ERROR COMPONENTES VECTOR

Se obtuvieron los datos en los ejes coordenados X , Y y Z tanto del Kinect como del láser, se realizó una diferencia entre ellos obteniendo de esta manera un error absoluto, que hace referencia al error presentado entre los datos obtenidos por un instrumento y los emitidos por el sensor.
Representa el porcentaje de error que hay de cada componente con respecto al dato real, se presenta debido a la variación de dato real a valor instrumental.

ERROR COMPONENTES VECTOR		
X	**Y**	**Z**
-0,0020	0,0019	0,0117
-0,0084	0,0143	0,0117
0,0065	0,0160	0,0267
0,0170	0,0107	0,0029
0,0015	0,0412	0,0336
0,0022	0,0009	-0,0164
-0,0036	0,0023	-0,0028
0,0149	0,0190	-0,0015
0,0041	0,0007	-0,0104
0,0063	0,0000	0,0129

0,0095	-0,0046	-0,0101
-0,0011	0,0221	0,0123
-0,0119	0,0220	-0,0031
-0,0077	-0,0001	0,0911
0,0112	-0,0013	-0,0218
0,22%	**0,40%**	**0,72%**

Tabla 5. Error componentes vector.

ERROR PROMEDIO JOINTS

Indica el promedio de error que hay en cada punto (contribución de error local con respecto a un conjunto general), se da debido a la ubicación del joint, algunos de ellos presentaron dificultad para ser medidos ya que el movimiento o el sentido de ubicación deformaban el esqueleto.

ERROR PROMEDIO JOINTS	**1,5016%**

ERROR PROMEDIO TOTAL

Se deja atrás los joints para realizar una visión global del error promedio como participación total del sistema analizado (esqueleto), representa el punto medio alcanzado como error en los datos.

ERROR PROMEDIO TOTAL	**0,45%**

ERROR TOTAL MAGNITUD

Error total del sistema analizado, hace referencia a un valor de tipo general donde se presentan la sumatoria de errores: de instrumento, de área de sensado, de precisión de medida, estabilidad de joint, entre otros.

Error total Magnitud	0,70%

Una vez realizados los errores, se procedió a realizar la desviación estándar y la media, utilizando las siguientes formulas:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

Ecuación 15. Desviación estándar.

$$\mu = \frac{x_1 + x_2 + \ldots + x_N}{N}$$

Ecuación 16. Media.

Datos requeridos para obtener la distribución Gaussiana, para los datos emitidos por el instrumento y real:

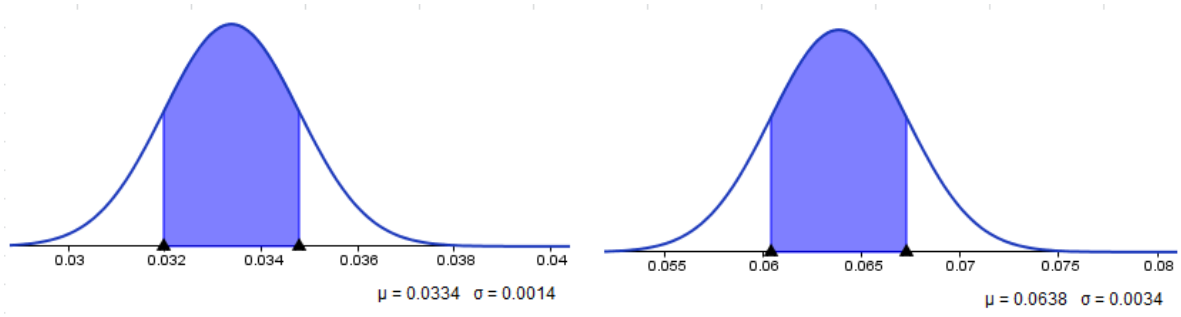


Figura 134. Distribución Gaussiana para el hombro.

La campana de Gauss es una función de probabilidad continua, cuyo máximo valor coincide con μ y dos puntos de inflexión situados en ambos extremos a una distancia σ

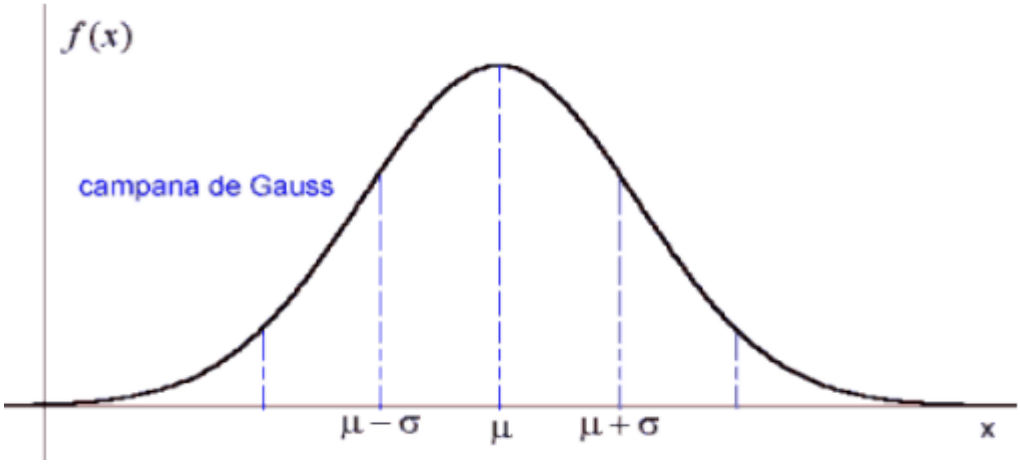


Figura 135. Parámetros de campana de Gauss [104].

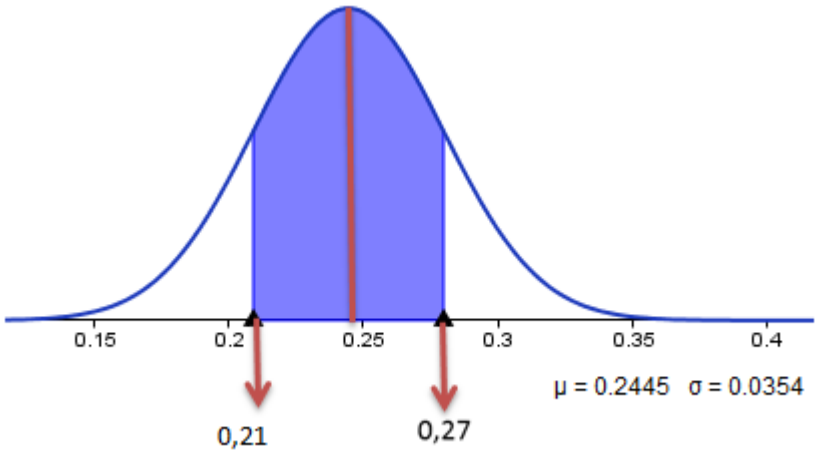


Figura 136. Valores para campana de Gauss del codo derecho.

Como se observa en la figura 136, la distribución gaussiana para el codo derecho, μ es de 0,24 cuyo máximo valor se evidencia en la campana y representa que la mayor parte de los datos están cercanos a este número , y los extremos están a una distancia σ (+- 0,0354). Lo que indica que existe una probabilidad de un 50% de observar un dato superior que la media, y un 50% de tener un dato inferior a la media.

La porción de la curva en color representa los valores en donde se concentran las medidas, intervalo de conjunto de datos [0,21 0,27] representa que el 68,26% de los datos recaen sobre esta zona.

El valor de la media (μ) influye de la siguiente manera: a mayor valor de μ la campana se desplaza hacia la derecha y a menor valor se acerca al origen coordenado.

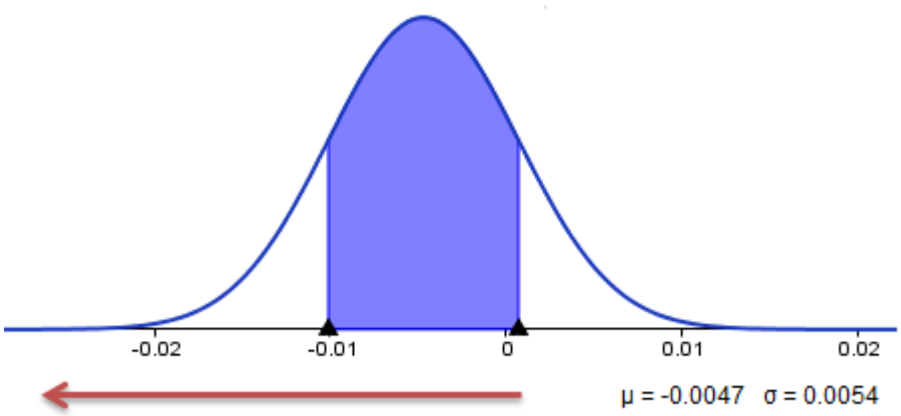


Figura 137. Valor de la media inferior a cero para el joint cadera izquierda.

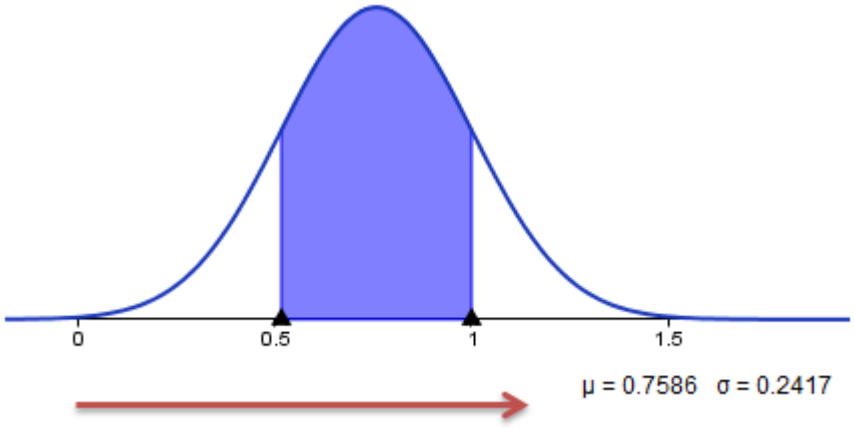


Figura 138. Valor de la media superior a cero para el joint cabeza.

La desviación estándar σ también tiene una influencia en la campana, es la que determina la forma a menor dispersión los datos se concentran alrededor de la media, y a mayor valor los datos se dispersan.

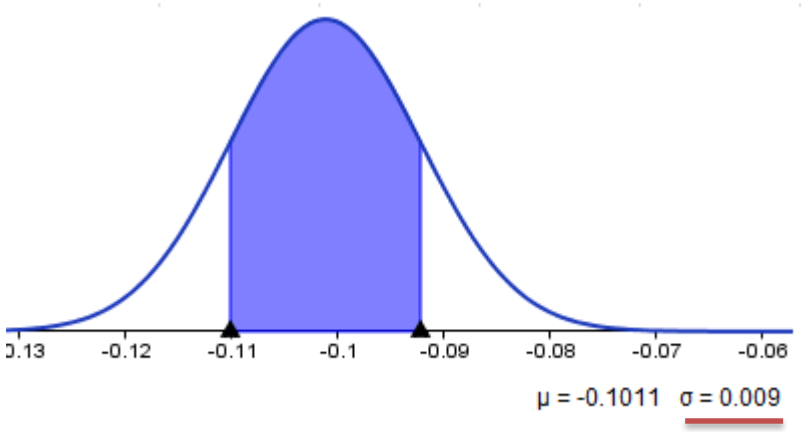
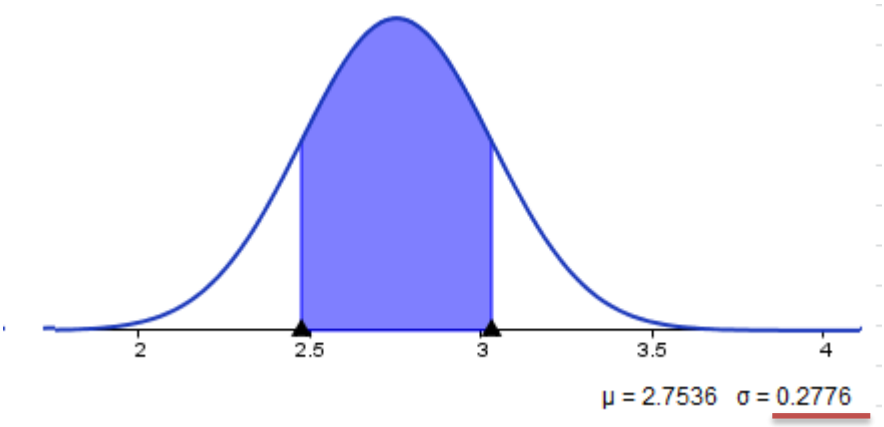


Figura 139. Comportamiento de los datos con desviación estándar pequeña.



Figura

Figura 140.Comportamiento de la desviación estándar de valor alto para el joint cuello.

En las figuras 139 y 140 se puede observar la concentración de datos cuando σ es pequeña, se acercan a μ y cuando σ es grande los valores tienden a alejarse. Tiene como objetivo trazar marcas de valores para determinar qué tan importantes son con referencia a la media.

Se utilizó esta distribución para determinar el área de concentración de los datos emitidos por el láser y el Kinect, visualizando si tienen un comportamiento similar. Ayuda a observar que tan alejado puede estar un valor teniendo como referencia μ, para establecer importancia en cuanto a medidas sobre paso del tiempo.

Estas graficas indican que tan alejados están los datos del eje coordenado, lo que indica que existen joints que pueden tener valores de comportamiento positivo y elevado por encima de otros puntos.

En general al utilizar esta herramienta se está haciendo un análisis de la concentración de los datos y su valor partiendo desde cero; además muestra que algunos joints tienen valores más estables y cercanos que otros.

CONCLUSIONES

El mayor error presentado es en el codo izquierdo, las rodillas y los pies, al tener el maniquí la mano flexionada el codo tuvo un grado de dificultad al momento de medir, ya que se realizaba una proyección del joint en un área que no era la del codo; la muñeca por tener los pies con cierta inclinación, el joint se posicionaba en una zona no apta y las rodillas poseían inestabilidad y el punto fluctuaba al momento de medir: Los menores se presentaron en la cabeza, el torso, la cadera y el cuello, debido a que estas zonas las identifica el controlador con estabilidad y sufren menos perturbaciones.

En el uso del Kinect como control de un robot, el usuario debe tener la precaución de no tomar poses que puedan generar errores irrespetando la captura total del cuerpo.

En el estudio realizado para la medición de las articulaciones del Kinect se utilizó dos paquetes comprendidos en ROS, el primero OpenNITracker el cual está incluido desde la versión de ROS llamada Hydro, tiene como ventaja de que genera la esqueletización de varios usuarios a los cuales les asigna un ID y solo cuando toman la posición Psi pose que establece las manos alzadas por encima de la cabeza, la cual se convierte en referencia y así genera la esqueletización, de esta forma se garantiza que el proceso se inicie de forma estable y evite posibles errores con detección de objetos que no sean cuerpos humanos o que se le asimilen, los ID son de nueva asignación cada vez que ingresa un usuario se realiza uno nuevo, esto hace que se cree un problema de seguridad ya que no habría un único operador de forma segura para el robot, porque se pueden presentar situaciones donde no se tenga la certeza de cuál es el usuario, obligando a reiniciar el proceso. El segundo paquete utilizado fue el Skeleton maker, el cual tiene la gran ventaja que solo transmite las articulaciones del primer usuario detectado y no genera una identificación, lo que garantiza que así ingresen más personas en la visual del Kinect los detecta como usuarios, los esqueletiza, pero no los transmite, solucionando el inconveniente de tener que reiniciar el proceso si un único usuario sale e ingresa de la visual del Kinect, pero esto se logra al no asignar el id del usuario en el esqueleto, esto se hace al tomar la pose de referencia frente a la cámara, en consecuencia para generar el esqueleto no es necesaria esta posición y se pueden presentar errores de esqueletización o inestabilidad en la misma. Pese a este último inconveniente es más práctico, seguro el Skeleton maker para la operación de un robot y el Openni Tracker es mejor como sistema de detección de personas cuando se requiera un monitoreo o explorar zonas o sistemas que requieran varios usuarios como operadores.

En el proceso de trasmisión de información entre nodos y paquetes, ROS dispone de mensajería, la cual se utilizó en esta aplicación usada para transmitir y recibir la información de las articulaciones del esqueleto generado, determinando el uso del paquete TF el cual crea una clase especifica con toda la información del esqueleto generado, donde se contiene tanto las coordenadas de cada articulación y su orientación en el espacio, de esta forma es más sencillo reconstruir esta información en otro paquete, pero lo más importante permite coordinar las articulaciones del operador con las articulaciones del robot, asignando una a una de acuerdo a las necesidades requeridas por el robot, situación que con solo mensajes es complicado.

A nivel de esqueletización la estabilidad presentada era de corto tiempo, lo que repercutía directamente sobre la medida, ya que si el esqueleto sufría una alteración la medida también se alteraba y se debía reiniciar el proceso. Se debe enfatizar en la imagen emitida por el controlador NITE ya que de esta depende que la visual del esqueleto se esté cumpliendo, estar pendiente de que los puntos articulados del cuerpo estén en posición correcta, lo que garantiza una medida confiable

TRABAJOS FUTUROS

ESTABILIDAD DEL KINECT

Desarrollar un mecanismo físico para empotrar, sobreponer o insertar el Kinect sobre alguna base modular con el objeto de evitar que se desajuste y produzca un error tanto en esqueletización como en medidas.

QUATERNION

Presentar un análisis aplicado a rotación y emisión de quaternion por parte del sensor; así como determinar su comportamiento en los diferentes joints con un usuario haciendo uso del Kinect.

TRATAMIENTO DEL SONIDO

Investigar sobre la emisión y transmisión del sonido en el Kinect aplicándolo a una problemática real; utilizando alguna técnica de tratamiento de sonido.

FILTRADO EN NUBE DE PUNTOS

Utilizando la librería PCL de ROS la cual emite una nube de puntos, implementar el filtrado haciendo uso de los filtros que existen en ROS, con el fin de realizar un tratamiento de imagen.

GAZEBO

Hacer uso de la herramienta de ROS, Gazebo para implementar un robot y hacer simulaciones con el mismo dependiendo de especificaciones requeridas y realizar un modelado del sensor Kinect con el fin de reducir el porcentaje de error.

BIBLIOGRAFIA

[1] CAUSA, Emiliano. Algoritmos de captura óptica de movimientos por substracción de video. Trabajo de grado (Ingeniero en sistemas). La Plata Argentina: Universidad Tecnológica Nacional. 2007, 16 p.

[2] RODRIGUEZ DIONISIO, Sandra Esperanza. Sistema de captura de movimiento grueso para animaciones tridimensionales en un ambiente controlado. Trabajo de grado (Ingeniera en sistemas). Bogotá D.C: Fundación Universitaria San Martin. Facultad de Ingenierías departamento de Sistemas, 2006, 63 p.

[3] RODRIGUEZ DIONISIO, Sandra Esperanza. Sistema de captura de movimiento grueso para animaciones tridimensionales en un ambiente controlado. Trabajo de grado (Ingeniera en sistemas). Bogotá D.C: Fundación Universitaria San Martin. Facultad de Ingenierías departamento de Sistemas, 2006, 63 p. Ibid., p 30.

[4] RODRIGUEZ DIONISIO, Sandra Esperanza. Sistema de captura de movimiento grueso para animaciones tridimensionales en un ambiente controlado. Trabajo de grado (Ingeniera en sistemas). Bogotá D.C: Fundación Universitaria San Martin. Facultad de Ingenierías departamento de Sistemas, 2006, 63 p. Ibid., p 30. Ibid., p 31

[5] GÓMEZ GARCIA, M. J. Sistema de captura de movimiento para el caminar humano. Análisis y comparativa. En: XIX Congreso Nacional de Ingeniería Mecánica (4: 20-22, Junio: Madrid, España). Memorias. Madrid: Asociación española de Ingeniería Mecánica, 2012. p. 8.

[6] GÓMEZ GARCIA, M. J. Sistema de captura de movimiento para el caminar humano. Análisis y comparativa. En: XIX Congreso Nacional de Ingeniería Mecánica (4: 20-22, Junio: Madrid, España). Memorias. Madrid: Asociación española de Ingeniería Mecánica, 2012. p. 8. Ibid., p. 6.

[7] VERA, Lucia. Espejo Aumentado: sistema interactivo de realidad aumentada basado en Kinect. En: Feria de turismo de Madrid (Fitur2011) (1: 10, Agosto: Madrid, España). Memorias. Madrid: 2011. p.10.

[8] VERA, Lucia. Espejo Aumentado: sistema interactivo de realidad aumentada basado en Kinect. En: Feria de turismo de Madrid (Fitur2011) (1: 10, Agosto: Madrid, España). Memorias. Madrid: 2011. p.10. Ibid., p.7.

[9] LOPEZ CONTRERAS, Oriol. Motion Capture amb Microsoft Kinect. Trabajo de grado (Ingeniero informático). Bellaterra España: Universidad Autónoma de Barcelona. Escuela de Ingeniería, 2012, 75 p.

[10] MARTÍNEZ, Helios de Rosario. Mejorar el equilibrio con video juegos. En: Proyecto StoppFalls Marco de la Unión Europea. (7: 10, Septiembre: Valencia, España). Memorias. Valencia: Instituto de Biomecánica de Valencia, grupo de tecnología sanitaria del IBV, 2012.p.4.

[11] MARTÍNEZ, Helios de Rosario. Mejorar el equilibrio con video juegos. En: Proyecto StoppFalls Marco de la Unión Europea. (7: 10, Septiembre: Valencia, España). Memorias. Valencia: Instituto de Biomecánica de Valencia, grupo de tecnología sanitaria del IBV, 2012.p.4. Ibid., p.3.

[12] LOMAS RODRÍGUEZ, Jorge. Estudio e integración de un sistema de control de entornos mediante gestos usando Kinect. Trabajo de grado (Ingeniero en informática). Madrid España: Universidad Autónoma de Madrid. Escuela Politécnica Superior. Ingeniería Informática, 2013, 102 p.

[13] GARRIDO DE LA FUENTE, David. Aplicaciones del Kinect para neurohabilitación. Trabajo de grado (Ingeniero en Telecomunicaciones).Catalunya España: Universidad politécnica de Catalunya. Escuela de ingeniería en Telecomunicaciones, 2012, 67 p.

[14] GARRIDO DE LA FUENTE, David. Aplicaciones del Kinect para neurohabilitación. Trabajo de grado (Ingeniero en Telecomunicaciones).Catalunya España: Universidad politécnica de Catalunya. Escuela de ingeniería en Telecomunicaciones, 2012, 67 p. Ibid,. p 68.

[15] RAMOS GUTIERREZ, Daniel. Estudio cinemático del cuerpo humano mediante Kinect. Trabajo de grado (Ingeniero en Telecomunicaciones). Madrid España: Universidad Politécnica de Madrid. Escuela técnica de telecomunicaciones, 2013, 118 p.

[16] MARTINEZ ZARZUELA, M. Monitorización del cuerpo humano en 3D mediante tecnología Kinect. En: Ministerio de ciencia e innovación (1: 15, Febrero: Valladolid España). Memorias. Departamento de teoría de la señal y Comunicaciones Universidad de Valladolid, 2011, 6 p.

[17] MARTINEZ ZARZUELA, M. Monitorización del cuerpo humano en 3D mediante tecnología Kinect. En: Ministerio de ciencia e innovación (1: 15, Febrero: Valladolid España). Memorias. Departamento de teoría de la señal y Comunicaciones Universidad de Valladolid, 2011, 6 p.
Ibid., p. 3.

[18] PARRILLA BEL, Luis. Interfaz gestual para el control de un robot humanoide con una cámara RGB-d. Trabajo de grado (Ingeniero en informática). Zaragoza España: Escuela de ingeniería y arquitectura. Departamento de informática e Ingeniería de sistemas, 2012, 82 p.

[19] GARCÍA LÓPEZ, Adrián. Estimación de zonas cruzables utilizando el sensor Kinect. Trabajo de grado (Grado en electrónica industrial y automática). Madrid España: Universidad Carlos III de Madrid. Ingeniería Electrónica industrial y automática, 2012, 67 p.

[20] BUSTAMANTE FILOZA, Iván Pablo. Diseño e implementación de software para neurorrehabilitación basada en detección de movimiento. Trabajo de grado (Ingeniero civil en computación). Santiago de Chile: Universidad de Chile. Facultad de ciencias físicas y matemáticas. Departamento de ciencias de la computación, 2011, 70 p.

[21] BUSTAMANTE FILOZA, Iván Pablo. Diseño e implementación de software para neurorrehabilitación basada en detección de movimiento. Trabajo de grado (Ingeniero civil

en computación). Santiago de Chile: Universidad de Chile. Facultad de ciencias físicas y matemáticas. Departamento de ciencias de la computación, 2011, 70 p. Ibid., p. 59.

[22] IBAÑEZ, Rodrigo. Herramienta para facilitar el desarrollo de las aplicaciones basadas e Kinect. Trabajo estudiantil. Buenos Aires Argentina: UNCPBA. Instituto de investigación ISISTAN. Facultad de ciencias exactas, 2013, 20 p.

[23] LÓPEZ MONRROY, Adrián Pastor y LEAL MELENDRES Jesús Adrián. Reconocimiento de gestos basado en modelos ocultos de Markov utilizando el Kinect. Investigación. Puebla México: Instituto nacional de astrofísica, óptica y electrónica, 2013. 10 p.

[24] LÓPEZ MONRROY, Adrián Pastor y LEAL MELENDRES Jesús Adrián. Reconocimiento de gestos basado en modelos ocultos de Markov utilizando el Kinect. Investigación. Puebla México: Instituto nacional de astrofísica, óptica y electrónica, 2013. 10 p. Ibid., p 8.

[25] AYALA CAJAS, CÉSAR ANDRÉS. Sistema de seguridad inteligente basado en reconocimiento de patrones mediante tecnología Kinect para restringir el acceso no autorizado a consolas de administración y monitoreo. Trabajo de grado (Ingeniero en sistemas e informática). Sangolquí Ecuador: Escuela politécnica del ejército. Departamento de ciencias de la computación. Carrera de Ingeniería de sistemas e informática, 2013, 159 p.

[26] CHUYA SUMBA, Jorge Patricio. Diseño e implementación de un sistema para el análisis del movimiento humano usando sensores Kinect. Trabajo de grado (Ingeniero electrónico). Cuenca Ecuador: Universidad Politécnica Salesiana. Carrera de Ingeniería Electrónica, 2013, 113 p.

[27] CHUYA SUMBA, Jorge Patricio. Diseño e implementación de un sistema para el análisis del movimiento humano usando sensores Kinect. Trabajo de grado (Ingeniero electrónico). Cuenca Ecuador: Universidad Politécnica Salesiana. Carrera de Ingeniería Electrónica, 2013, 113 p. Ibid., p 29.

[28] CAMARGO C, Esperanza. Sistema portátil de captura de movimiento para el análisis sistemático de la marcha humana. En: Red de revistas científicas de América Latina, El Caribe, España y Portugal (16: 34, Octubre- Diciembre: Bogotá D.C). Memorias. Bogotá D.C: Sistema de información Científica, 2012. p. 18.

[29] CAMARGO C, Esperanza. Sistema portátil de captura de movimiento para el análisis sistemático de la marcha humana. En: Red de revistas científicas de América Latina, El Caribe, España y Portugal (16: 34, Octubre- Diciembre: Bogotá D.C). Memorias. Bogotá D.C: Sistema de información Científica, 2012. p. 18. Ibid., p. 15.

[30] VILLA GARZÓN, Diego Andrés. Manipulación inalámbrica de un brazo robótico por medio del Kinect. Armenia Quindío: Universidad del Quindío. Programa de tecnología en Instrumentación Electrónica. Grupo GIDECT, 2013, 6 p.

[31] MUÑOZ CARDONA, John. Sistema de rehabilitación basado en el uso de análisis biomecánico y videojuegos mediante el sensor Kinect. En: Tecno.Lógicas (Edición

especial, Octubre: Pereira, Risaralda). Memorias. Pereira: Maestría en Ingeniería Eléctrica, Grupo de investigación y desarrollo en cultura de la salud, 2013. p. 13.

[32] MUÑOZ CARDONA, John. Sistema de rehabilitación basado en el uso de análisis biomecánico y videojuegos mediante el sensor Kinect. En: Tecno.Lógicas (Edición especial, Octubre: Pereira, Risaralda). Memorias. Pereira: Maestría en Ingeniería Eléctrica, Grupo de investigación y desarrollo en cultura de la salud, 2013. p. 13. Ibid., p. 9.

[33] CASILLAS ALCALÁ, Paul. Un tabletop basada en Kinect para manipulación de objetos virtuales 2D y 3D. Trabajo de grado (Maestro en ciencias de computación). México DF: Centro de investigación y de estudios avanzados del Instituto politécnico nacional. Unidad Zacatenco. Departamento de Computación, 2012, 159 p.

[34] CASILLAS ALCALÁ, Paul. Un tabletop basada en Kinect para manipulación de objetos virtuales 2D y 3D. Trabajo de grado (Maestro en ciencias de computación). México DF: Centro de investigación y de estudios avanzados del Instituto politécnico nacional. Unidad Zacatenco. Departamento de Computación, 2012, 159 p. Ibid., p. 50-51.

[35] PERALTA BENHUMEA, Sergio Herman. Interfaz de lenguaje natural usando Kinect. Trabajo de grado (Maestro en ciencias en computación). México D.F: Centro de investigación y de estudios avanzados del instituto politécnico nacional. Unidad de Zacatenco. Departamento de computación, 2012, 110 p.

[36] PERALTA BENHUMEA, Sergio Herman. Interfaz de lenguaje natural usando Kinect. Trabajo de grado (Maestro en ciencias en computación). México D.F: Centro de investigación y de estudios avanzados del instituto politécnico nacional. Unidad de Zacatenco. Departamento de computación, 2012, 110 p. Ibid., p33.

[37] PERALTA BENHUMEA, Sergio Herman. Interfaz de lenguaje natural usando Kinect. Trabajo de grado (Maestro en ciencias en computación). México D.F: Centro de investigación y de estudios avanzados del instituto politécnico nacional. Unidad de Zacatenco. Departamento de computación, 2012, 110 p. Ibid., p34.

[38] VÁSQUEZ VÁSQUEZ, Francisco Javier. Brazo robótico controlado mediante sensor Kinect. Trabajo de grado (Ingeniero en comunicaciones y electrónica). México D.F: Instituto politécnico nacional. Escuela superior de Ingeniería Mecánica y Eléctrica, 2013, 138 p.

[39] VÁSQUEZ VÁSQUEZ, Francisco Javier. Brazo robótico controlado mediante sensor Kinect. Trabajo de grado (Ingeniero en comunicaciones y electrónica). México D.F: Instituto politécnico nacional. Escuela superior de Ingeniería Mecánica y Eléctrica, 2013, 138 p. Ibid., p31.

[40] VÁSQUEZ VÁSQUEZ, Francisco Javier. Brazo robótico controlado mediante sensor Kinect. Trabajo de grado (Ingeniero en comunicaciones y electrónica). México D.F: Instituto politécnico nacional. Escuela superior de Ingeniería Mecánica y Eléctrica, 2013, 138 p. Ibid., p31.

[41] VÁSQUEZ VÁSQUEZ, Francisco Javier. Brazo robótico controlado mediante sensor Kinect. Trabajo de grado (Ingeniero en comunicaciones y electrónica). México D.F:

Instituto politécnico nacional. Escuela superior de Ingeniería Mecánica y Eléctrica, 2013, 138 p. Ibid., p32.

[42] VÁSQUEZ VÁSQUEZ, Francisco Javier. Brazo robótico controlado mediante sensor Kinect. Trabajo de grado (Ingeniero en comunicaciones y electrónica). México D.F: Instituto politécnico nacional. Escuela superior de Ingeniería Mecánica y Eléctrica, 2013, 138 p. Ibid., p32.

[43] VÁSQUEZ VÁSQUEZ, Francisco Javier. Brazo robótico controlado mediante sensor Kinect. Trabajo de grado (Ingeniero en comunicaciones y electrónica). México D.F: Instituto politécnico nacional. Escuela superior de Ingeniería Mecánica y Eléctrica, 2013, 138 p. Ibid., p34.

[44] VÁSQUEZ VÁSQUEZ, Francisco Javier. Brazo robótico controlado mediante sensor Kinect. Trabajo de grado (Ingeniero en comunicaciones y electrónica). México D.F: Instituto politécnico nacional. Escuela superior de Ingeniería Mecánica y Eléctrica, 2013, 138 p. Ibid., p35.

[45] PERALTA BENHUMEA, Sergio Herman. Interfaz de lenguaje natural usando Kinect. Trabajo de grado (Maestro en ciencias en computación). México D.F: Centro de investigación y de estudios avanzados del instituto politécnico nacional. Unidad de Zacatenco. Departamento de computación, 2012, 110 p.Op. Cit., p. 37.

[46] PERALTA BENHUMEA, Sergio Herman. Interfaz de lenguaje natural usando Kinect. Trabajo de grado (Maestro en ciencias en computación). México D.F: Centro de investigación y de estudios avanzados del instituto politécnico nacional. Unidad de Zacatenco. Departamento de computación, 2012, 110 p. Op. Cit., p.38.

[47] PERALTA BENHUMEA, Sergio Herman. Interfaz de lenguaje natural usando Kinect. Trabajo de grado (Maestro en ciencias en computación). México D.F: Centro de investigación y de estudios avanzados del instituto politécnico nacional. Unidad de Zacatenco. Departamento de computación, 2012, 110 p. Op. Cit., p.38.

[48] BOLAÑOS, Felipe. Dejando Huella Grafiti Digital. Trabajo de grado. Quito Ecuador: Universidad San Francisco de Quito. Colegio de comunicación y artes contemporáneas, 2012, 32.

[49] ILVAY TADAY, RÓMULO BYRON. Sistema de educación para niños de 3 a 5 años, mediante un robot controlado por el sensor Kinect. Trabajo de grado (Ingeniero electrónico, control y redes industriales). Escuela Superior Politécnica de Chimborazo. Facultad de informática y electrónica. Escuela de ingeniería electrónica en control y redes industriales. Riobamba Ecuador: 2014, 139 p.

[50] ILVAY TADAY, RÓMULO BYRON. Sistema de educación para niños de 3 a 5 años, mediante un robot controlado por el sensor Kinect. Trabajo de grado (Ingeniero electrónico, control y redes industriales). Escuela Superior Politécnica de Chimborazo. Facultad de informática y electrónica. Escuela de ingeniería electrónica en control y redes industriales. Riobamba Ecuador: 2014, 139 p. Ibid., p140.

[51] ILVAY TADAY, RÓMULO BYRON. Sistema de educación para niños de 3 a 5 años, mediante un robot controlado por el sensor Kinect. Trabajo de grado (Ingeniero

electrónico, control y redes industriales). Escuela Superior Politécnica de Chimborazo. Facultad de informática y electrónica. Escuela de ingeniería electrónica en control y redes industriales. Riobamba Ecuador: 2014, 139 p.Ibid., p140.

[52] VIÑALS PONS, Joaquín. Localización y generación de mapas del entorno (SLAM) de un robot por medio de una Kinect. Trabajo de grado (Ingeniero en informática). Escola Técnica Superior d'Enginyeria Informática. Universidad Politécnica de Valencia. Valencia España: 2012, 99 p.

[53] CORRAL MARTÍN, José María. Colores y Sombras. Universidad de salamanca. Ingeniería superior informática. Informática gráfica. Salamanca España.

[54] VIÑALS PONS, Joaquín. Localización y generación de mapas del entorno (SLAM) de un robot por medio de una Kinect. Trabajo de grado (Ingeniero en informática). Escola Técnica Superior d'Enginyeria Informática. Universidad Politécnica de Valencia. Valencia España: 2012, 99 p Ibid., p100.

[55] VIÑALS PONS, Joaquín. Localización y generación de mapas del entorno (SLAM) de un robot por medio de una Kinect. Trabajo de grado (Ingeniero en informática). Escola Técnica Superior d'Enginyeria Informática. Universidad Politécnica de Valencia. Valencia España: 2012, 99 p Ibid., p102.

[56] RODRÍGUEZ DIONICIO, Sandra Esperanza. Sistema de captura de movimiento grueso para animaciones tridimensionales en un ambiente controlado. Trabajo de grado. Bogotá D.C: Fundación universitario San Martín. Facultad de ingeniería. Departamento de sistemas, 2006, 63 p.

[57] RODRÍGUEZ DIONICIO, Sandra Esperanza. Sistema de captura de movimiento grueso para animaciones tridimensionales en un ambiente controlado. Trabajo de grado. Bogotá D.C: Fundación universitario San Martín. Facultad de ingeniería. Departamento de sistemas, 2006, 63 p. Ibid., p64.

[58] PANTRIGO FERNÁNDEZ, Juan José. Análisis biomecánico del movimiento humano mediante técnicas de visión artificial. España: ESCET-URJC

[59] URBINA ROJAS, Wilson Fabián. Diseño e implementación de un sistema de captura de movimientos de las extremidades del cuerpo humano. Trabajo de grado (ingeniero en Control). Bogotá D.C: Universidad Distrital Francisco José de Caldas. Ingeniería en Control, 2012, 21 p.

[60] URBINA ROJAS, Wilson Fabián. Diseño e implementación de un sistema de captura de movimientos de las extremidades del cuerpo humano. Trabajo de grado (ingeniero en Control). Bogotá D.C: Universidad Distrital Francisco José de Caldas. Ingeniería en Control, 2012, 21 p. Ibid., p11.

[61] URBINA ROJAS, Wilson Fabián. Diseño e implementación de un sistema de captura de movimientos de las extremidades del cuerpo humano. Trabajo de grado (ingeniero en Control). Bogotá D.C: Universidad Distrital Francisco José de Caldas. Ingeniería en Control, 2012, 21 p. Ibid., p12.

[62] URBINA ROJAS, Wilson Fabián. Diseño e implementación de un sistema de captura de movimientos de las extremidades del cuerpo humano. Trabajo de grado (ingeniero en Control). Bogotá D.C: Universidad Distrital Francisco José de Caldas. Ingeniería en Control, 2012, 21 p. Ibid., p13

[63] URBINA ROJAS, Wilson Fabián. Diseño e implementación de un sistema de captura de movimientos de las extremidades del cuerpo humano. Trabajo de grado (ingeniero en Control). Bogotá D.C: Universidad Distrital Francisco José de Caldas. Ingeniería en Control, 2012, 21 p. Ibid., p13

[64] URBINA ROJAS, Wilson Fabián. Diseño e implementación de un sistema de captura de movimientos de las extremidades del cuerpo humano. Trabajo de grado (ingeniero en Control). Bogotá D.C: Universidad Distrital Francisco José de Caldas. Ingeniería en Control, 2012, 21 p. Ibid., p14.

[65] CAUSA, Emiliano. Algoritmos de captura de movimiento por substracción de video. Una implementación en sintaxis de Processing (Java). Buenos Aires Argentina: Grupo de investigación Biopus, 2013, 16 p.

[66] CAUSA, Emiliano. Algoritmos de captura de movimiento por substracción de video. Una implementación en sintaxis de Processing (Java). Buenos Aires Argentina: Grupo de investigación Biopus, 2013, 16 p. Ibid., p17.

[67] CAUSA, Emiliano. Algoritmos de captura de movimiento por substracción de video. Una implementación en sintaxis de Processing (Java). Buenos Aires Argentina: Grupo de investigación Biopus, 2013, 16 p. Ibid., p18.

[68] RANGEL DE LA TORRE, Ricardo. Modelado, control y simulación de un quadrotor equipado con un brazo manipulado robótico. Trabajo de grado (ingeniero Industrial). Sevilla España: Universidad de Sevilla. Departamento de Ingeniería en sistemas y automática, 2012, 306 p.

[69] RANGEL DE LA TORRE, Ricardo. Modelado, control y simulación de un quadrotor equipado con un brazo manipulado robótico. Trabajo de grado (ingeniero Industrial). Sevilla España: Universidad de Sevilla. Departamento de Ingeniería en sistemas y automática, 2012, 306 p. Ibid., p92.

[70] RANGEL DE LA TORRE, Ricardo. Modelado, control y simulación de un quadrotor equipado con un brazo manipulado robótico. Trabajo de grado (ingeniero Industrial). Sevilla España: Universidad de Sevilla. Departamento de Ingeniería en sistemas y automática, 2012, 306 p. Ibid., p94.

[71] RANGEL DE LA TORRE, Ricardo. Modelado, control y simulación de un quadrotor equipado con un brazo manipulado robótico. Trabajo de grado (ingeniero Industrial). Sevilla España: Universidad de Sevilla. Departamento de Ingeniería en sistemas y automática, 2012, 306 p. Ibid., p94.

[72] RANGEL DE LA TORRE, Ricardo. Modelado, control y simulación de un quadrotor equipado con un brazo manipulado robótico. Trabajo de grado (ingeniero Industrial). Sevilla España: Universidad de Sevilla. Departamento de Ingeniería en sistemas y automática, 2012, 306 p. Ibid., p95.

[73] RANGEL DE LA TORRE, Ricardo. Modelado, control y simulación de un quadrotor equipado con un brazo manipulado robótico. Trabajo de grado (ingeniero Industrial). Sevilla España: Universidad de Sevilla. Departamento de Ingeniería en sistemas y automática, 2012, 306 p. Ibid., p98.

[74] RANGEL DE LA TORRE, Ricardo. Modelado, control y simulación de un quadrotor equipado con un brazo manipulado robótico. Trabajo de grado (ingeniero Industrial). Sevilla España: Universidad de Sevilla. Departamento de Ingeniería en sistemas y automática, 2012, 306 p. Ibid., p100.

[75] RANGEL DE LA TORRE, Ricardo. Modelado, control y simulación de un quadrotor equipado con un brazo manipulado robótico. Trabajo de grado (ingeniero Industrial). Sevilla España: Universidad de Sevilla. Departamento de Ingeniería en sistemas y automática, 2012, 306. Ibid., p101.

[76] MORGAN QUIGLEY Y JOSH FAUST. Ros.h File Reference [En línea]. Disponible en <http://docs.ros.org/indigo/api/roscpp/html/ros_8h.html> [Consulta Octubre de 2014]

[77] KEN CONLEY Y MORGAN QUIGLEY. Package.h File Reference [En línea]. Disponible en <http://docs.ros.org/hydro/api/roslib/html/c++/package_8h.html> [Consulta en Octubre de 2014]

[78] KEN CONLEY Y MORGAN QUIGLEY. Package.h File Reference [En línea]. Disponible en <http://docs.ros.org/hydro/api/roslib/html/c++/package_8h.html> [Consulta en Octubre de 2014]

[79] Disponible en internet: KEN CONLEY Y MORGAN QUIGLEY. Package.h File Reference [En línea]. Disponible en <http://docs.ros.org/hydro/api/roslib/html/c++/package_8h.html> [Consulta en Octubre de 2014]

[80] TULLY FOOTE Y EITAN MARDER-EPPSTEIN. Tf::TransformBroadcaster Class Reference [En línea]. Disponible en <http://docs.ros.org/indigo/api/tf/html/c++/classtf_1_1TransformBroadcaster.html> [Consulta en Octubre de 2014]

[81] TULLY FOOTE Y EITAN MARDER-EPPSTEIN. Tf::TransformBroadcaster Class Reference [En línea]. Disponible en <http://docs.ros.org/indigo/api/tf/html/c++/classtf_1_1TransformBroadcaster.html> [Consulta en Octubre de 2014]

[82] MANUEL FERNÁNDEZ CARMONA. Api de OpenNI para hmc [En Línea]. Disponible en <http://ocw.unia.es/ciencias-tecnologicas/tecnologia-del-ocio/materiales-basicos-folder/html/B1_UD05/api_de_openni_para_hmc.html/skinless_view> [Consulta Noviembre de 2014]

[83] SHOTTON, Jamie. Real-Time Human Pose recognition in Parts from Single Depth Images. Microsoft Research Cambridge & Xbox Incubation.

[84] SHOTTON, Jamie. Real-Time Human Pose recognition in Parts from Single Depth Images. Microsoft Research Cambridge & Xbox Incubation.

[85] SHOTTON, Jamie. Real-Time Human Pose recognition in Parts from Single Depth Images. Microsoft Research Cambridge & Xbox Incubation.

[86] ARIÑO SUSTAETA, Jorge. Desarrollo de un robot de servicio para la asistencia a personas de la tercera edad. Trabajo de grado (Ingeniería Informática). Valencia España: Universidad politécnica de Valencia. Escuela técnica superior de ingeniería Informática, 2014. 105.

[87] DANIEL STONIER. Tf [En Línea]. Disponible en:< http://wiki.ros.org/tf> [Consulta Noviembre de 2014]

[88] WOPFNER Manuel, Managing Coordinate frame Transformations in Distributed, Component- Based Robot System. University of Applied Sciences U1M. Department of Computer Science. Master Course Information Systems. 2011, 72 p.

[89] WOPFNER Manuel, Managing Coordinate frame Transformations in Distributed, Component- Based Robot System. University of Applied Sciences U1M. Department of Computer Science. Master Course Information Systems. 2011, 72 p. Ibid., p13.

[90] WOPFNER Manuel, Managing Coordinate frame Transformations in Distributed, Component- Based Robot System. University of Applied Sciences U1M. Department of Computer Science. Master Course Information Systems. 2011, 72 p. Ibid., p14.

[91] WOPFNER Manuel, Managing Coordinate frame Transformations in Distributed, Component- Based Robot System. University of Applied Sciences U1M. Department of Computer Science. Master Course Information Systems. 2011, 72 p. Ibid., p14.

[92] LUCAS WALTER. Rotations in ROS [En Línea]. Disponible en:< http://wiki.ros.org/geometry/RotationMethods> [Consulta Noviembre de 2014]

[93] GAVILÁN, María Elena, Simulación por dinámica molecular del movimiento de un trompo pesado. En: revista colombiana de física, Abril, 2006. Vol. 38, No 1, pág. 417-419.

[94] GAVILÁN, María Elena, Simulación por dinámica molecular del movimiento de un trompo pesado. En: revista colombiana de física, Abril, 2006. Vol. 38, No 1, pág. 417-419. Ibid., p 102.

[95] GAVILÁN, María Elena, Simulación por dinámica molecular del movimiento de un trompo pesado. En: revista colombiana de física, Abril, 2006. Vol. 38, No 1, pág. 417-419. Ibid., p102.

[96] GAVILÁN, María Elena, Simulación por dinámica molecular del movimiento de un trompo pesado. En: revista colombiana de física, Abril, 2006. Vol. 38, No 1, pág. 417-419. Ibid., p103.

[97] GAVILÁN, María Elena, Simulación por dinámica molecular del movimiento de un trompo pesado. En: revista colombiana de física, Abril, 2006. Vol. 38, No 1, pág. 417-419. Ibid., p103.

[98] TORRES DEL CASTILLO, G.F. La representación de rotaciones mediante cuaterniones. Trabajo de grado. Puebla México: Universidad Autónoma de Puebla. Instituto de ciencias, departamento de física matemática, 1999, 8 p.

[99] TORRES DEL CASTILLO, G.F. La representación de rotaciones mediante cuaterniones. Trabajo de grado. Puebla México: Universidad Autónoma de Puebla. Instituto de ciencias, departamento de física matemática, 1999, 8 p. Ibid., p2.

[100] TORRES DEL CASTILLO, G.F. La representación de rotaciones mediante cuaterniones. Trabajo de grado. Puebla México: Universidad Autónoma de Puebla. Instituto de ciencias, departamento de física matemática, 1999, 8 p. Ibid., p2.

[101] TORRES DEL CASTILLO, G.F. La representación de rotaciones mediante cuaterniones. Trabajo de grado. Puebla México: Universidad Autónoma de Puebla. Instituto de ciencias, departamento de física matemática, 1999, 8 p. Ibid., p3.

[102] SKIL EUROPE. Medidor de distancias por láser [En línea]. Disponible en:< http://www.skileurope.com/es/es/diyocs/herramientas/1208/508/medicion-de-distancias/medidor-de-distancias-por-laser/> [Consulta Noviembre de 2014]

[103] SKIL EUROPE. Medidor de distancias por láser [En línea]. Disponible en:< http://www.skileurope.com/es/es/diyocs/herramientas/1206/1143/nivelador-laser/dispositivo-de-nivelacion-laser/> [Consulta Noviembre de 2014]

[104] RD LOZANO. Los errores [En Línea]. Disponible en: < http://www.coloryapariencia.com.ar/errores.htm> [Consulta Enero de 2015]

ANEXOS

Anexo 1. Código para lectura de Joints (100 muestras) y su promedio

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <cmath>



float x_suma;
float y_suma;
float z_suma;
float x_prom;
float y_prom;
float z_prom;
float x_cien[99];
float y_cien[100];
float z_cien[100];
float qx;
float qy;
float qz;
float qw;


//matrix [fila][columna]
char
frames[16][20]={"/openni_depth_frame","/head","/neck","/torso","/left_shoulder","/left_elbo
w","/left_hand","/right_shoulder","/right_elbow","/right_hand","/left_hip","/left_knee","/left_fo
ot","/right_hip","/right_knee","/right_foot"} ;

int main( int argc, char** argv )
{ int i=0;

  ros::init(argc, argv, "skeleton");
  ros::NodeHandle n,node;


  tf::TransformListener listener;
  listener.waitForTransform("/openni_depth_frame", "/head", ros::Time(),
ros::Duration(5.0));


 tf::StampedTransform transform;
  ros::Rate r(10);
```

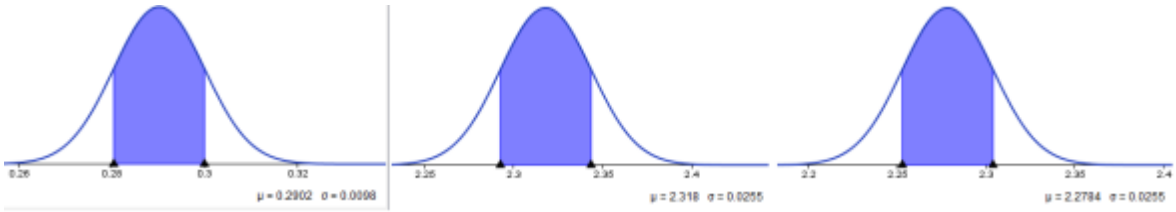
```
while (node.ok())
{


        ROS_INFO("coordenada x frame_id= %s", frames[9]);
          for(i=0;i<=99;i++)

                                {       try{
                                        listener.lookupTransform(frames[0], frames[9],
                                ros::Time(0), transform);
                                        }
                                        catch (tf::TransformException ex)
                                        {  ROS_ERROR("%s",ex.what());}

                                        x_cien[i]=transform.getOrigin().x();
                                        y_cien[i]=transform.getOrigin().y();
                                        z_cien[i]=transform.getOrigin().z();
                                        //qx[i]=transform.getRotation().qx();
                                        //qx[i]=transform.getRotation().qx();
                                        //qx[i]=transform.getRotation().qx();

                                        if(i == 0){
                                        x_suma=x_cien[0];
                                        y_suma=y_cien[0];
                                        z_suma=z_cien[0];

                                        }
                                        else{
                                        x_suma=x_cien[i]+x_suma;
                                        y_suma=y_cien[i]+y_suma;
                                        z_suma=z_cien[i]+z_suma;
                                        }
                                        qx=transform.getRotation().x();
                                        qy=transform.getRotation().y();
                                        qz=transform.getRotation().z();
                                        qw=transform.getRotation().w();



                ros::Duration(0.05).sleep();
                                }

                x_prom=x_suma/(i);
                y_prom=y_suma/(i);
                z_prom=z_suma/(i);

                ROS_INFO("datos x=%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
```

```
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f", x_cien[0], x_cien[1], x_cien[2], x_cien[3], x_cien[4], x_cien[5], x_cien[6],
x_cien[7], x_cien[8], x_cien[9], x_cien[10], x_cien[11], x_cien[12], x_cien[13], x_cien[14],
x_cien[15], x_cien[16], x_cien[17], x_cien[18], x_cien[19], x_cien[20], x_cien[21],
x_cien[22], x_cien[23], x_cien[24], x_cien[25], x_cien[26], x_cien[27], x_cien[28],
x_cien[29], x_cien[30], x_cien[31], x_cien[32], x_cien[33], x_cien[34], x_cien[35],
x_cien[36], x_cien[37], x_cien[38], x_cien[39], x_cien[40], x_cien[41], x_cien[42],
x_cien[43], x_cien[44], x_cien[45], x_cien[46], x_cien[47], x_cien[48], x_cien[49],
x_cien[50], x_cien[51], x_cien[52], x_cien[53], x_cien[54], x_cien[55], x_cien[56],
x_cien[57], x_cien[58], x_cien[59], x_cien[60], x_cien[61], x_cien[62], x_cien[63],
x_cien[64], x_cien[65], x_cien[66], x_cien[67], x_cien[68], x_cien[69], x_cien[70],
x_cien[71], x_cien[72], x_cien[73], x_cien[74], x_cien[75], x_cien[76], x_cien[77],
x_cien[78], x_cien[79], x_cien[80], x_cien[81], x_cien[82], x_cien[83], x_cien[84],
x_cien[85], x_cien[86], x_cien[87], x_cien[88], x_cien[89], x_cien[90],
x_cien[91],x_cien[92], x_cien[93], x_cien[94], x_cien[95], x_cien[96], x_cien[97],
x_cien[98], x_cien[99]);

                    ROS_INFO("datos y=%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f  %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f", y_cien[0], y_cien[1], y_cien[2], y_cien[3], y_cien[4], y_cien[5], y_cien[6],
y_cien[7], y_cien[8], y_cien[9], y_cien[10], y_cien[11], y_cien[12], y_cien[13], y_cien[14],
y_cien[15], y_cien[16], y_cien[17], y_cien[18], y_cien[19], y_cien[20], y_cien[21],
y_cien[22], y_cien[23], y_cien[24], y_cien[25], y_cien[26], y_cien[27], y_cien[28],
y_cien[29], y_cien[30], y_cien[31], y_cien[32], y_cien[33], y_cien[34], y_cien[35],
y_cien[36], y_cien[37], y_cien[38], y_cien[39], y_cien[40], y_cien[41], y_cien[42],
y_cien[43], y_cien[44], y_cien[45], y_cien[46], y_cien[47], y_cien[48], y_cien[49],
y_cien[50], y_cien[51], y_cien[52], y_cien[53], y_cien[54], y_cien[55], y_cien[56],
y_cien[57], y_cien[58], y_cien[59], y_cien[60], y_cien[61], y_cien[62], y_cien[63],
y_cien[64], y_cien[65], y_cien[66], y_cien[67], y_cien[68], y_cien[69], y_cien[70],
y_cien[71], y_cien[72], y_cien[73], y_cien[74], y_cien[75], y_cien[76], y_cien[77],
y_cien[78], y_cien[79], y_cien[80], y_cien[81], y_cien[82], y_cien[83], y_cien[84],
y_cien[85], y_cien[86], y_cien[87], y_cien[88], y_cien[89], y_cien[90], y_cien[91],
y_cien[92], y_cien[93], y_cien[94], y_cien[95], y_cien[96], y_cien[97], y_cien[98],
y_cien[99]);

                    ROS_INFO("datos z=%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f %f %f %f %f %f %f %f %f  %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f", z_cien[0], z_cien[1], z_cien[2], z_cien[3], z_cien[4], z_cien[5], z_cien[6],
z_cien[7], z_cien[8], z_cien[9], z_cien[10], z_cien[11], z_cien[12], z_cien[13], z_cien[14],
z_cien[15], z_cien[16], z_cien[17], z_cien[18], z_cien[19], z_cien[20], z_cien[21],
z_cien[22], z_cien[23], z_cien[24], z_cien[25], z_cien[26], z_cien[27], z_cien[28],
z_cien[29], z_cien[30], z_cien[31], z_cien[32], z_cien[33], z_cien[34], z_cien[35],
z_cien[36], z_cien[37], z_cien[38], z_cien[39], z_cien[40], z_cien[41], z_cien[42],
z_cien[43], z_cien[44], z_cien[45], z_cien[46], z_cien[47], z_cien[48], z_cien[49],
z_cien[50], z_cien[51], z_cien[52], z_cien[53], z_cien[54], z_cien[55], z_cien[56],
```

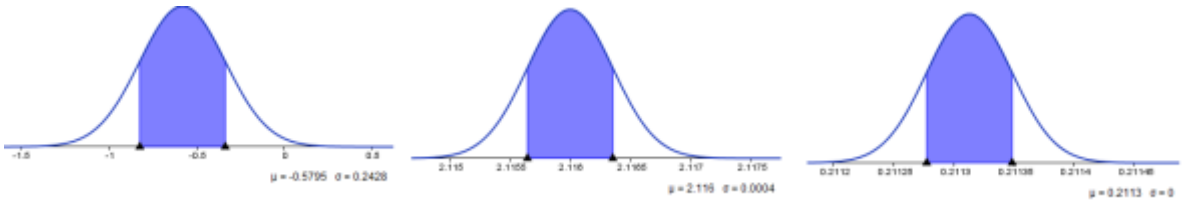
```
z_cien[57], z_cien[58], z_cien[59], z_cien[60], z_cien[61], z_cien[62], z_cien[63],
z_cien[64], z_cien[65], z_cien[66], z_cien[67], z_cien[68], z_cien[69], z_cien[70],
z_cien[71], z_cien[72], z_cien[73], z_cien[74], z_cien[75], z_cien[76], z_cien[77],
z_cien[78], z_cien[79], z_cien[80], z_cien[81], z_cien[82], z_cien[83], z_cien[84],
z_cien[85], z_cien[86], z_cien[87], z_cien[88], z_cien[89], z_cien[90], z_cien[91],
z_cien[92], z_cien[93], z_cien[94], z_cien[95], z_cien[96], z_cien[97], z_cien[98],
z_cien[99]);

                ROS_INFO("suma x =%f suma y =%f suma z=%f ", x_suma, y_suma,
z_suma);
                ROS_INFO("promedio x=%f y=%f z=%f ", x_prom, y_prom, z_prom);
                ROS_INFO("quaternion qx=%f qy=%f qz=%f  qw=%f", qx, qy, qz, qw);


 }

}
```

Anexo 2. Gaussianas para datos reales

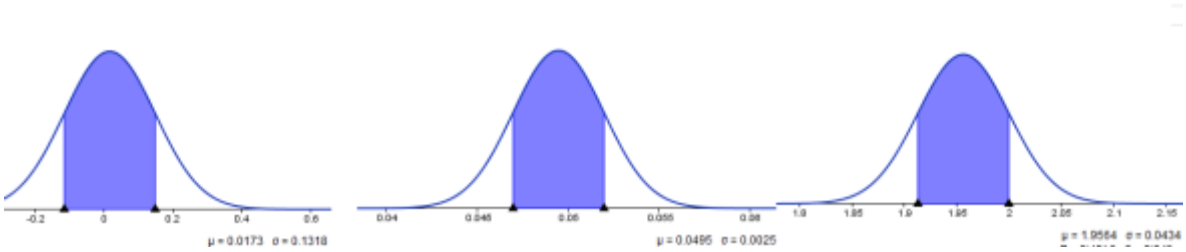
Codo Izquierdo



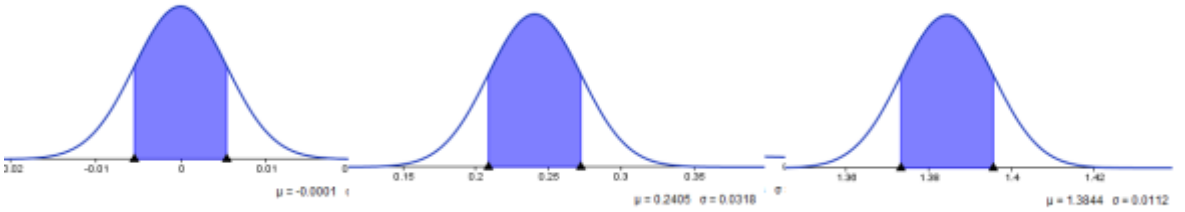
Torso



Hombro izquierdo

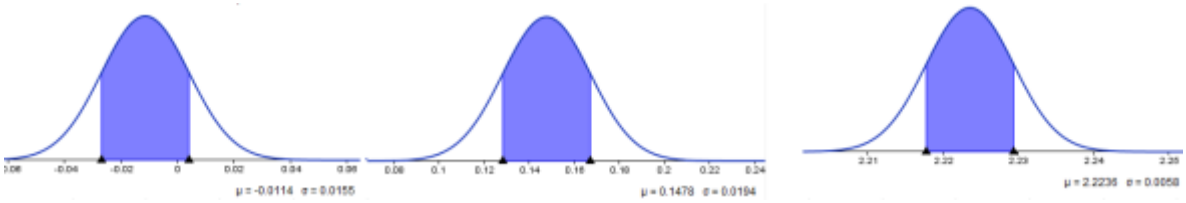


Cadera Izquierda

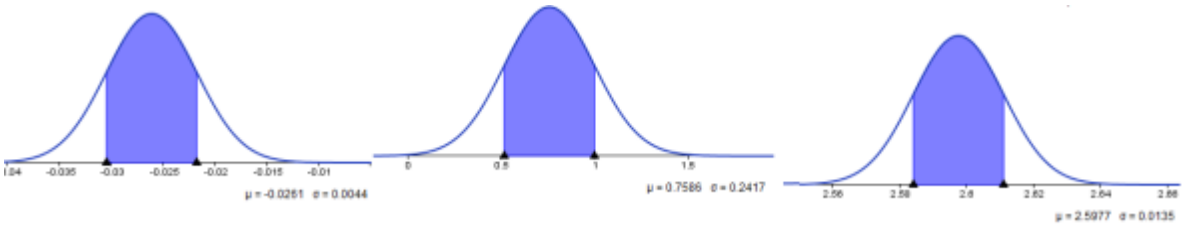


Mano izquierda

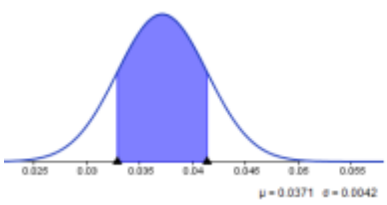
Cadera derecha

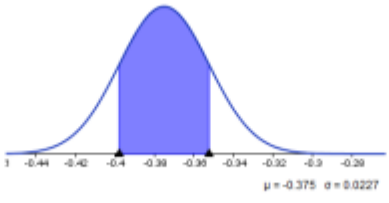


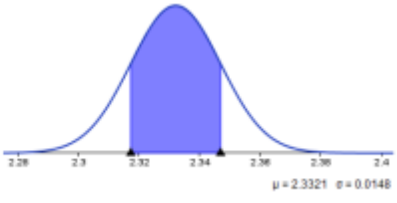
Cabeza



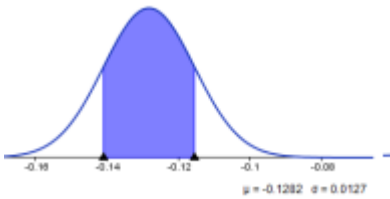
Rodilla derecha

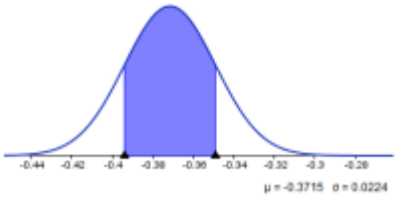


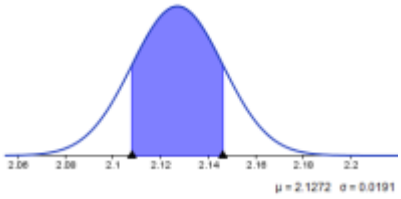




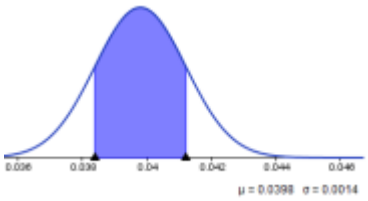
Rodilla izquierda

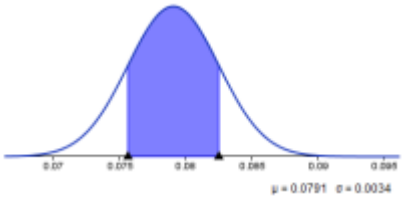


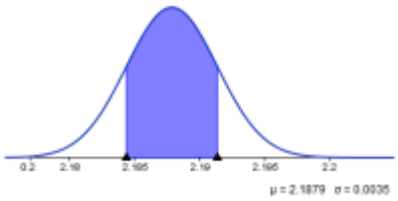




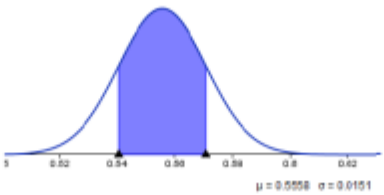
Hombro derecho



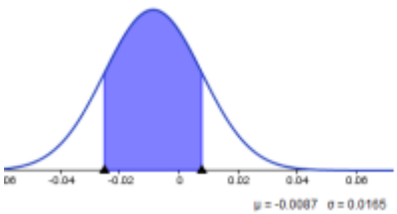


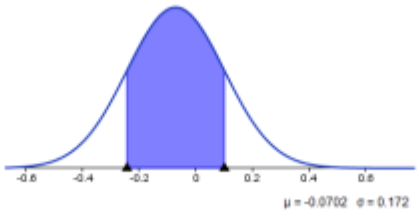


Cuello

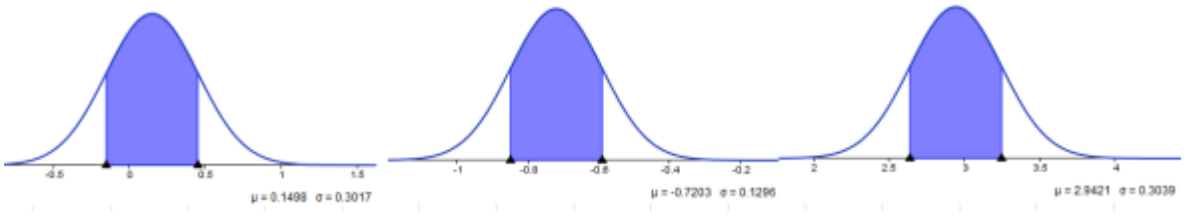


Pie derecho



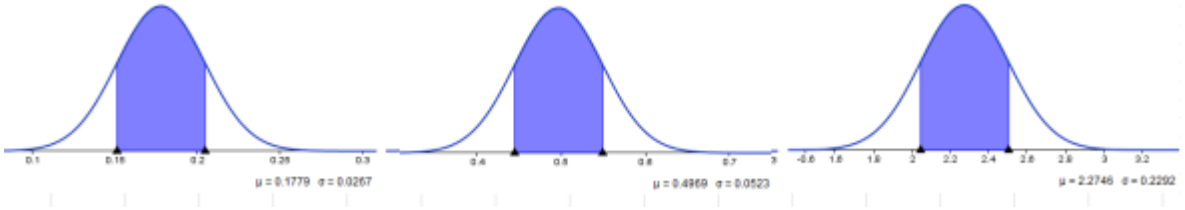


Pie izquierdo

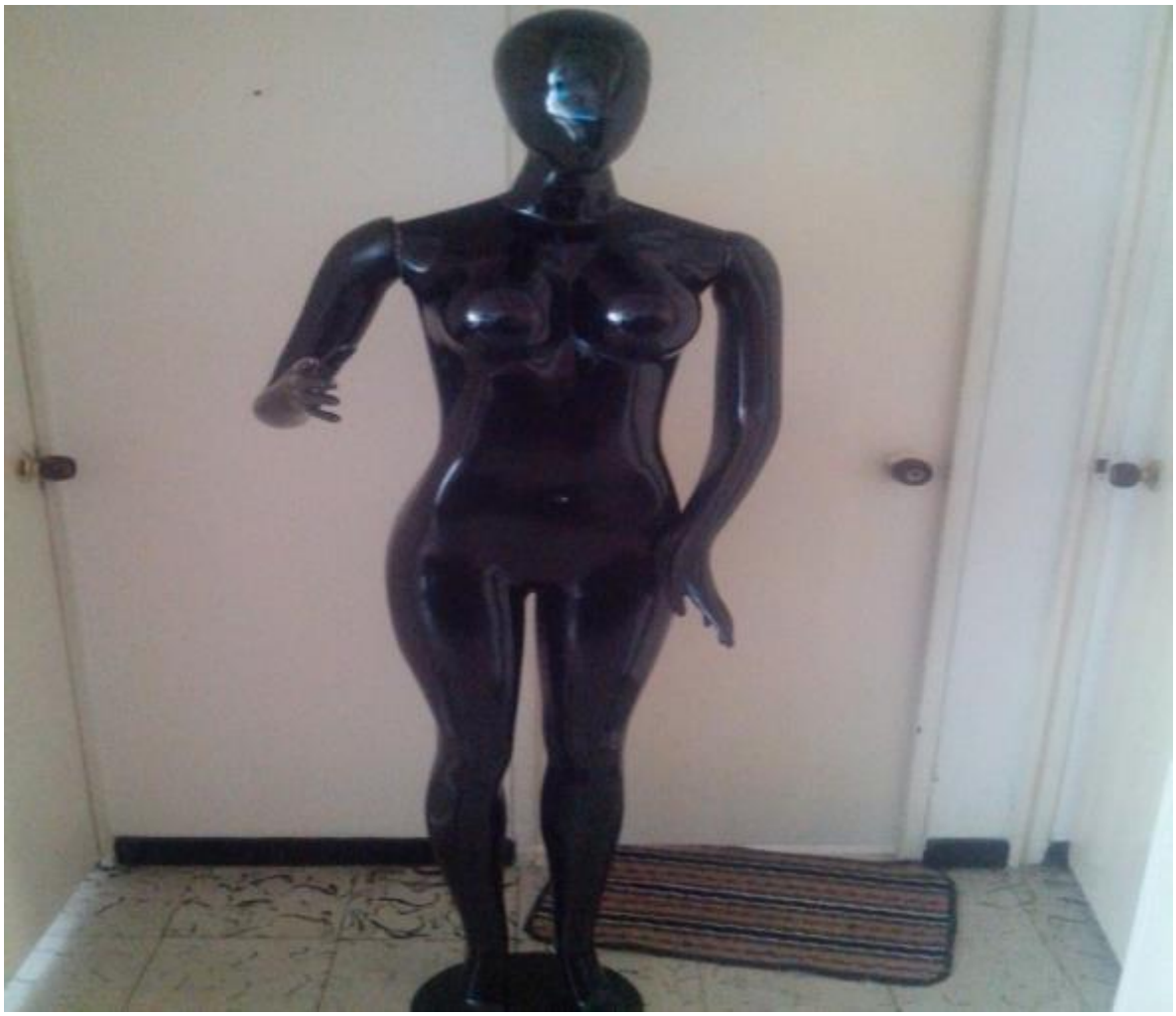


Codo derecho

Mano derecha



Anexo 3. Maniquí utilizado



Altura: 1.70 cm.

Talla: 8

Contorno del pecho: 84 cm

Contorno de cintura: 68 cm

Contorno de cadera: 90 cm.