Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Desarrollo de un sistema de visión 3D para su integración en un robot móvil social.

Autor:

Juárez Sánchez, Alexis

Tutores:

Gómez García-Bermejo, Jaime

Zalama Casanova, Eduardo

Ingeniería de Sistemas y Automática

Valladolid, Diciembre de 2014.

Resumen

En una rama de la ingeniería tan importante como la robótica, con dispositivos cada vez en más estrecha relación con el mundo que nos rodea, se ha hecho muy necesaria la ayuda de sistemas de interacción hombre-máquina adecuados para abordar esta relación de la forma más natural posible. La visión artificial es uno de los campos que más apoyo ofrece en el desarrollo de estos sistemas de interacción.

Este proyecto aborda la comprensión, diseño e implementación de un sistema de visión artificial 3D con el objetivo de controlar e interactuar con un robot móvil de tipo social.

Palabras clave: Visión artificial, robótica, robot social, Kinect, interacción hombre-máquina.

Abstract

In such an important engineering branch like robotics, with devices increasingly in contact with the world that surrounds us, it has been necessary the helping of human-machine interaction systems in order to tackle this relationship in the most natural way. Computer vision is one of the fields that offers more support in the development of these interaction systems.

This project approaches the comprehension, design and implementation of a 3D computer vision system with the objective of controlling and interacting with a social mobile robot.

Keywords: computer vision, robotics, social robot, Kinect, human-machine interaction.

Índice

Capítulo 1: Planteamiento inicial 11
1.1 Introducción 11
1.2 Objetivos. 12
1.3 Descripción de la memoria 14

Capítulo 2: Interacción hombre-máquina y robótica social 17
2.1 Interacción hombre-máquina. 17
2.2 Interacción mediante visión 3D 19
2.3 Robótica social y robótica de servicios. 22

Capítulo 3: Análisis de requerimientos 27
3.1 Análisis preliminar. 27
3.1.1 Análisis del robot y su emplazamiento. 27
3.1.2 Análisis de las funciones del robot 27
3.2 Sistemas de Visión Artificial 31
3.3 Kinect como sistema de VA. 34

Capítulo 4: Programación basada en componentes. ROS 37
4.1 Introducción. Sistemas operativos para robots. 37
4.2 Conceptos básicos de ROS. 39
4.2.1 Modelo editor-suscriptor y paso de mensajes. 39
4.2.2 Versiones, instalación y robots soportados. 40
4.3 Programación en ROS. 42
4.3.1 Conceptos avanzados: TF y nubes de puntos 43
4.3.2 Creación de paquetes y ejecutables. 45
4.3.3 Herramientas de depuración y visualización. 46

Capítulo 5: Los robots Sacarino y Edubot 49
5.1 Introducción a los robots. 50
5.2 Especificaciones físicas de los robots 53
5.2.1 Plataforma mecánica móvil 53
5.2.2 Sensores 56
5.2.3 Estructura de control 57
5.2.4 Ubicación de Kinect 57
5.3 Software y programación 60
5.3.1 Control a alto nivel 60
5.3.2 Sistema operativo 62

Capítulo 6: Diseño y desarrollo de los sistemas de reconocimiento 65
6.1 Configuración inicial. 65
6.2 Sistema de reconocimiento de personas. 67
6.2.1 Inicio y calibración. 67
6.2.2 Modelo del usuario 69
6.3 Sistema de reconocimiento gestual. 72
6.3.1 Inicio y adquisición de datos. 72
6.3.2 Interpretación de gestos. 73
6.3.3 Aplicación de los gestos. 77
6.4 Sistema de seguimiento. 78
6.4.1 Inicio y lectura de la posición del usuario. 78
6.4.2 Seguimiento. 80
6.4.3 Gestos y acciones en el robot. 80
6.5 Integración y descripción general de la aplicación 83
6.6 Configuración y ejecución de la aplicación. 85

Capítulo 7: Resultados y validación experimental 87
7.1 Diseño de experimentos. 87
7.2 Ubicación óptima de Kinect. 80
7.3 Resultados y consecución de objetivos. 94
7.4 Pruebas y verificación de las condiciones de operación. 99
7.4.1 Sistema de reconocimiento de personas. 99
7.4.2 Sistema de reconocimiento gestual. 100
7.4.3 Sistema de seguimiento. 102

Capítulo 8: Estudio económico 105
8.1 Recursos empleados. 105
8.2 Costes directos. 106
8.2.1 Costes del personal. 106
8.2.2 Costes de amortización de equipos y programas 108
8.2.3 Costes derivados de otros materiales 109
8.2.4 Costes directos totales 110
8.3 Costes indirectos 110
8.4 Costes totales 111

Capítulo 9: Conclusión y futuros desarrollos 113

Bibliografía y referencias 117

Anexo: Hoja de características de Kinect 121

Índice de figuras y tablas

Figura 2.1. UAV con cámara.....19
Figura 2.2. Google Car y mapa 3D generado.....21
Figura 2.3. Reconocimiento de objetos. 22
Figura 2.4. ASIMO. 23
Figura 2.5. A.L.O. Botlr, robot asistente en hoteles.....25
Figura 3.1. Entorno ejemplo para el robot. 28
Figura 3.2. Juego de realidad aumentada. 30
Figura 3.3. Kinect. 32
Figura 3.4. Xtion PRO.....32
Figura 3.5. Leap Motion. 33
Figura 3.6. Láser Riegl.....33
Figura 3.7. Diagrama de funcionamiento de la calibración de Kinect. 35
Figura 3.8. Puntos de luz proyectados por el sensor IR. 35
Figura 3.9. Rangos de visión en Kinect.....36
Figura 4.1. Esquema ejemplo de módulos en ROS.....39
Figura 4.2. Ejemplos de robots que incorporan ROS. 42
Figura 4.3. TF marcados sobre las articulaciones de un robot. 44
Figura 4.4. Tratamiento de una nube de puntos con PCL. 45
Figura 4.5. rqt_console. 47
Figura 4.6. rqt_graph. 47
Figura 4.7. Programa de visualización Rviz. 48
Figura 4.8. Ejemplo de pdf generado por view_frames 48
Figura 5.1. Parte superior de Sacarino.....50
Figura 5.2. Parte inferior de Sacarino 50
Figura 5.3. Cara de Sacarino 2. 51
Figura 5.4. Kinect incorporado en el chasis del robot. 52
Figura 5.5. Parte inferior anterior de Sacarino 2.....52
Figura 5. 6. Plataforma móvil Edubot.....54
Figura 5.7. Plataforma móvil de Sacarino 2. 54
Figura 5.8. Sistema sincrodrive de Sacarino. 55

Figura 5.9. Disposición cinemática de Sacarino 2. 55
Figura 5.10. Plataforma móvil de Sacarino 2. 57
Figura 5.11. Controladora GPMRC. 58
Figura 5.12. Arquitectura de Edubot. 59
Figura 5.13. Joystick de tipo PS3. 61
Figura 5.14. Nodos preexistentes en el robot. 63
Figura 6.1. Nodos del robot preexistentes. 66
Figura 6.2. Pose Psy. 68
Figura 6.3. Calibración de usuarios. 69
Figura 6.4. Árbol de transformed frames. 70
Figura 6.5. Mensaje de tipo tf. 71
Figura 6.6. Distribución espacial de los tf. 71
Figura 6.7. Coordenadas de los tf. 73
Figura 6.8. Brazo izquierdo en cruz. 75
Figura 6.9. Brazo derecho en cruz. 75
Figura 6.10. Brazos en cruz. 76
Figura 6.11. Stop. 76
Figura 6.12. Brazos en alto. 76
Figura 6.13. Diagrama de flujo simplificado. 78
Figura 6.14. Mensaje de tipo joy. 81
Figura 6.15. Grafo de nodos y topics de los sistemas. 84
Figura 7.1. Turtlesim. 88
Figura 7.2. Edubot. 88
Figura 7.3. Sacarino. 89
Figura 7.4. Mapa 2D del entorno de pruebas. 90
Figura 7.5. Entorno de pruebas. 91
Figura 7.6. Entorno real. 91
Figura 7.7. Medidas de las alturas de Kinect en Sacarino 2. 92
Tabla 7.1. Campo de visión vertical de Kinect. 93
Figura 7.8. Kinect en su implementación final. 93
Figura 7.9. Reconocimiento de usuario principal y secundarios. 96
Figura 7.10. Vista trasera desde Kinect. 97
Figura 7.11. Vista trasera externa. 97

Figura 7.12. Vista acercamiento desde Kinect.....98
Figura 7.13. Vista acercamiento externa.....98
Figura 7.14. Vista stop desde Kinect.....98
Figura 7.15. Vista stop externa.....98
Figura 7.16. Falso positivo en la cámara.....100
Figura 7.17. Kinect en posición vertical.....101
Figura 7.18. Tracker inclinado.....101
Tabla 7.2. Coordenadas erróneas con el tracker torcido.....101
Figura 7.19. Manos fuera de rango.....102
Tabla 8.1. Salarios.....107
Tabla 8.2. Días de trabajo.....107
Tabla 8.3. Distribución temporal de trabajo.....108
Tabla 8.4. Amortización de material.....109
Tabla 8.5. Coste del material consumible.....110
Tabla 8.6. Costes indirectos.....110
Tabla 8.7. Costes totales.....111

Capítulo 1:

Planteamiento inicial

1.1 Introducción.

El presente proyecto se enmarca dentro de los campos de la robótica y de la visión artificial, ambos estrechamente unidos debido a la importancia de un adecuado sistema de percepción visual en los robots de hoy en día.

La robótica es una de las ramas más recientes de la tecnología, desarrollándose principalmente en la segunda mitad del siglo XX, y que por tanto, aún está en constante expansión. Por otro lado, la visión artificial es un campo muy asociado a la robótica, pero no exclusivo de ella, puesto que tiene otros muchos usos. Ambos campos dependen sobremanera de las tecnologías desarrolladas para los mismos, razón por la cual no ha sido hasta las últimas décadas cuando han evolucionado notablemente

En concreto, las nuevas plataformas y tecnologías que han surgido en los últimos años han acercado estos campos a un público más amplio debido a la mayor facilidad de uso de algunas de estas nuevas tecnologías y al abaratamiento de costes de las susodichas.

En robótica se pueden distinguir dos grandes campos: la robótica industrial y la robótica de servicios. Los robots de tipo industrial son aquellos que se encuentran en grandes fábricas o empresas, y que por lo general se encargan

de realizar o ayudar a la realización de procesos industriales. Por el contrario, los robots de servicios se encargan de asistir y ayudar a los humanos en diversas tareas, generalmente no enfocadas al ámbito industrial.

Por tanto, este proyecto estará enmarcado más concretamente dentro de la robótica de servicios, debido a la importancia en ella de los sistemas de interacción hombre-máquina, especialmente aquellos mediante visión artificial, como los que este trabajo aborda.

A su vez, dentro de la robótica de servicios, debido a la gran cantidad de aplicaciones que engloba, se pueden encontrar robots de muchos tipos, cómo robots biomiméticos, humanoides, plataformas móviles… siendo cada uno de ellos más adecuado para unas tareas u otras. En este caso, los robots a los que se espera vaya destinado el sistema desarrollado aquí pueden ser muy variados, pero deben contar todos ellos con una plataforma móvil que les dote de movimiento y preferiblemente una estructura que asemeje una forma humana. En cualquier caso, la principal función que se espera controlar mediante la visión artificial es la interacción y la movilidad del propio robot, por lo que es su diseño como plataforma móvil lo que más interesa aquí.

Al hilo de lo mencionado anteriormente, sobre que el robot final se asemeje a una persona real, hay que destacar la relevancia de ello en la robótica actual. Si se desean incorporar robots de servicios realizando labores de asistencia personal, guía, o en definitiva, cualquier trabajo en contacto con humanos, es importante que dicho contacto sea lo más natural posible, y en ello, tanto el aspecto del robot como los mecanismos de interacción con el mismo son de suma importancia.

Por último, aunque el sistema de visión artificial se puede utilizar en prácticamente cualquier tipo de robot, los objetivos primordiales y en los que se espera la implementación y pruebas iniciales son los robots desarrollados por el centro tecnológico Cartif. Cabe destacar que este trabajo se apoya para sus pruebas en 2 proyectos realizados con anterioridad: por un lado, un proyecto de diseño de una plataforma robótica móvil con objeto de realizar sobre ella otros trabajos de investigación y/o aprendizaje, y en la que se harán las primeras pruebas sobre movimiento; por otro lado, un desarrollo previo de Cartif de un robot de características y objetivos concretos (asistencia en hoteles), del que se desea realizar una segunda iteración mejorando algunos aspectos del mismo, entre ellos la interacción mediante visión artificial mencionada. Todo esto condiciona algunos aspectos del trabajo actual, que sin embargo no limitan su utilidad, sino que ayudan a definirlo con más claridad.

1.2 Objetivos.

A la hora de realizar este proyecto, hay que tener en cuenta varios aspectos:

- El uso que se le va a dar al robot, en este caso la interacción social, y más concretamente la interacción en un entorno amplio y sencillo con usuarios en principio ajenos al funcionamiento del robot.

- Las limitaciones que puedan existir en la plataforma física construida con anterioridad y que haga uso del dispositivo de visión artificial.

- La plataforma software preelegida con anterioridad, en el caso de existir, como sucede en este caso, y que limitará algunas acciones o posibilidades de desarrollo.

Con todo ello, se espera buscar una utilidad real al robot para su desempeño en el entorno en el que se pretende introducir, teniendo en cuenta las ventajas e inconvenientes del diseño del mismo, buscando potenciar las primeras y superando en la medida de lo posible los segundos, de forma que además se apliquen algunos de los últimos avances en interacción y visión artificial en robótica.

Así, para enfocar adecuadamente el proyecto, se ha elegido como objetivo principal el desarrollo de un sistema de interacción hombre-máquina mediante visión artificial 3D en el que el robot responda a los estímulos, preferentemente humanos, de su entorno de forma adecuada.

Para realizar esto, se han fijado una serie de objetivos intermedios, gracias a los cuales se llegará poco a poco a la solución adoptada finalmente. Éstos objetivos son:

- Elección de un dispositivo adecuado para la interacción mediante visión artificial, teniendo en cuenta las características hardware y software del robot previamente establecidas.

- Programación del reconocimiento de personas y la interacción con el entorno de acuerdo a los estímulos externos recibidos.

- Realización de todo lo anterior con la mejor relación posible entre prestaciones y coste.

El segundo de los objetivos se detallará en un capítulo posterior para definir más claramente la forma de interacción entre el robot y su entorno de entre las muchas posibilidades que se pueden abordar.

1.3 Descripción de la memoria.

Una vez definidos los objetivos del proyecto, a la hora de abordar la memoria y la documentación del mismo, se puede seguir un orden y unos apartados bien diferenciados, que se corresponderán con los diferentes capítulos del presente texto.

En el primer capítulo, el presente, se ha realizado una introducción a la robótica y su relación con los sistemas de interacción y más concretamente de visión artificial. Asimismo se ha definido el objetivo principal y los objetivos secundarios de todo el proyecto.

En el segundo capítulo se hace un recorrido por el estado actual, tanto de la robótica social, principal ámbito de aplicación de este proyecto, como de la interacción hombre-máquina, totalmente necesaria para el desempeño de la primera, haciendo un especial hincapié en sistemas de visión artificial en tres dimensiones.

En el tercer capítulo se realiza el análisis en detalle de los requerimientos y se especifican los objetivos concretos a abordar en el trabajo, tanto respecto a las funciones a realizar mediante el sistema de visión artificial como la elección del dispositivo físico concreto a utilizar.

El capítulo cuatro describe la programación basada en componentes, que es la que se va a utilizar aquí, y especialmente el sistema operativo ROS, en el que se va a programar todo, con el objetivo de entender la estructuración de la implementación realizada posteriormente.

En el capítulo quinto se describen brevemente los sistemas que pueden hacer uso de la tecnología desarrollada en este proyecto, y se explican en detalle las plataformas de pruebas utilizadas. Asimismo, se describen los requerimientos respecto a la plataforma de control y otros elementos necesarios en el robot, y las interacciones entre las diferentes partes del mismo, tanto hardware como software.

El capítulo seis es el núcleo del presente trabajo. En él se describe completamente y al detalle el diseño y desarrollo de las funcionalidades con visión artificial previamente establecidas, explicando el porqué de algunos

detalles y elecciones de diseño. También se expone el modo de funcionamiento de los sistemas desarrollados por separado, o actuando completamente el conjunto de los mismos a la vez, incluyendo un manual de uso y lanzamiento de los programas.

En el séptimo capítulo se hace una relación clara de los resultados obtenidos en los programas anteriores en los varios robots que se han utilizado, las pruebas realizadas para la colocación adecuada del sistema de visión artificial, así como su fiabilidad, condiciones óptimas para el uso y una comprobación del grado de cumplimiento de los objetivos del proyecto.

El octavo capítulo consiste en un estudio del coste económico de todo el proyecto, tratando en detalle los costes del sistema de visión artificial desarrollado en este proyecto, los costes de personal o de otros materiales y recursos utilizados en la elaboración del mismo, con el objetivo de ver la viabilidad económica del mismo y determinar si se cumple el objetivo relacionado con ella.

Para terminar, en el capítulo noveno se hace una recapitulación general del proyecto desarrollado en su totalidad para enumerar las conclusiones obtenidas, determinar lo aprendido y las dificultades encontradas así como proponer posibles mejoras y ampliaciones del mismo.

Después, para concluir la memoria se lista la bibliografía consultada y referenciada a lo largo del texto y los anexos, que incluyen las especificaciones técnicas del hardware utilizado.

Capítulo 2:

Interacción hombre-máquina y robótica social

En primer lugar, previo a la realización del sistema objeto del presente proyecto, se ha realizado un estudio del estado actual de las tecnologías similares a la abordada con el fin de conocer las soluciones propuestas por otros equipos de investigación o empresas para resolver problemáticas similares a las aquí planteadas.

Para ello, se observarán dos campos principales: por un lado la interacción hombre-máquina y los métodos más avanzados para realizar dicha interacción, y por otro lado la situación de la robótica social y los robots de servicios en la actualidad, a fin de averiguar qué tipos de robots semejantes a aquel con el que se pretende trabajar existen y sus funciones principales.

2.1 Interacción hombre-máquina.

Desde que existen dispositivos electrónicos, ha sido labor principal de diseñadores y programadores facilitar el intercambio de información entre las personas y estos dispositivos. Haciendo uso de los principales sentidos del ser humano, estas interfaces han sido siempre capaces de entregar

información principalmente de carácter visual y auditivo. Esto no es complicado de conseguir, puesto que se tienen pantallas y altavoces capaces de entregar esta información de manera fácil desde hace mucho tiempo.

No obstante, es más complicado diseñar dispositivos capaces de leer e interpretar esta información en lugar de simplemente entregarla. Y sin embargo, es un paso necesario para que la robótica alcance una posición importante y los robots interactúen con el mundo real. Por ello, desde hace décadas, se desarrollan sensores, la mayoría inspirados en la propia naturaleza, que tratan de leer la información del mundo exterior de la manera más fidedigna posible, muchas veces de manera similar a cómo lo hacen las propias personas o animales.

Algunas de las tecnologías de sensorización que permiten tomar datos del exterior y, con un procesamiento posterior, extraer conclusiones del mundo real, son muy rudimentarias, como por ejemplo los sensores por contacto, o por radiación, mediante ultrasonidos o luz. Estos métodos han sido y aún siguen siendo muy importantes para determinados robots, facilitando enormemente su comprensión del entorno, en especial del lugar en el que se encuentran de forma barata.

Pero se quedan cortos a la hora de facilitar la interacción con humanos, que requiere de un gran volumen de datos, velocidad y capacidad de análisis de los mismos para que la interacción se realice de forma natural.

En esto ha ayudado en gran medida la visión artificial. Es cierto que podemos considerar los haces de luz o los sistemas láser en 2D como un sistema de “visión” simplificado, pero es la captación de imágenes en dos y tres dimensiones lo que de verdad ha hecho y está haciendo evolucionar la forma de interactuar de los robots con los seres humanos y otros elementos del entorno de forma significativa.

La visión artificial es un campo de la inteligencia artificial y de la robótica que lleva décadas entre nosotros, pero que por desgracia no contaba con la tecnología suficiente para manejar el alto volumen de datos que necesita para funcionar correctamente, tanto en imágenes 2D cómo en 3D. Esto ha cambiado en la actualidad, y las cámaras, el tratamiento y el procesamiento de imágenes se encuentran en muchos dispositivos de uso diario.

Sin ir más lejos, la irrupción de los Smartphone en la vida cotidiana ha popularizado el uso no sólo de cámaras, sino de tecnologías y algoritmos que antes eran sólo de uso profesional. Las cámaras de los móviles de hoy en día cuentan con tratamiento de imágenes de todo tipo, y con tecnologías de reconocimiento muy variadas, como la detección de sonrisas o el

reconocimiento de patrones, figuras y formas. La mayoría de los usos de estas tecnologías son de tipo lúdico, pero no dejan de ser formas de aprovecharlas en usos cotidianos que antes no se podías ni imaginar.

Otros dispositivos que cada vez se están volviendo más comunes son los drones o UAVs (Unmanned Aerial Vehicle), vehículos aéreos de pequeño tamaño usados en investigación, en ocio u otros campos (rescate, operaciones militares, etc.) y que en su práctica totalidad cuentan con cámaras o sistemas de visión artificial (*figura 2.1*) más o menos sofisticados que les permiten realizar diferentes tareas: ayudar a la propia navegación, inspección e interacción con el entorno, o de nuevo, aplicaciones de realidad aumentada.

Figura 2.1. UAV con cámara.

Pero las cámaras y sistemas de visión artificial no se quedan sólo en aparatos caros aunque de alcance general. Con la popularización de aparatos y plataformas de desarrollo Open Source, que acercan la robótica al gran público, han aparecido sistemas y cámaras de bajo coste para dotar de visión artificial a pequeños proyectos sin gastar cantidades desorbitadas, facilitando además la investigación, muy importante para el desarrollo de otras tecnologías. Este es el caso de la Raspberry Pi y su módulo de cámara, que en conjunto puede dotar de un complejo sistema de visión a robots y pequeños proyectos de aficionados.

2.2 Interacción mediante visión 3D.

Los métodos para la recolección de imágenes en tres dimensiones, esto es, que tengan en cuenta la profundidad, son muy variados, clasificándose en los siguientes tipos:

- Métodos mediante visión monocular: son métodos muy sencillos que utilizan imágenes 2D y el uso de la iluminación en ellas para determinar distancias y formas geométricas muy simples. Usa técnicas como el cálculo geométrico, la detección de esquinas de Harris o el campo aleatorio de Markov (MRF) para obtener información de las imágenes y extraer algunas características 3D, pero muy elementales [27].

- Métodos *Time-of-flight* (tiempo de vuelo): son los más antiguos y conocidos, pero excesivamente caros. Las cámaras tiempo de vuelo pueden captar la profundidad de una escena y sus elementos [7] gracias a un rayo que se proyecta y recibe, y en base al tiempo entre uno y otro se calcula la distancia de cada punto.

- Métodos estereoscópicos: son métodos que usan dos cámaras distintas que, simulando la visión binocular humana, son capaces de crear imágenes en tres dimensiones mediante la combinación de dos imágenes en dos dimensiones [9].

- Métodos de luz estructurada: usan una cámara RGB convencional junto a un emisor y receptor de infrarrojos que permite calcular las distancias de los puntos proyectados mediante triangulación. Las cámaras que usan este tipo de visión se denominan RGB-D [8].

De los anteriores, los más comunes y en los que se desarrollan más aplicaciones y tecnologías en la actualidad son los dos últimos, puesto que son métodos fiables y baratos que poco a poco se han ido incorporando en más tecnologías, tanto en grupos de investigación como en empresas de tecnología.

Por lo general son las grandes empresas y algunos organismos públicos los que tienen el capital para llevar a cabo grandes inversiones en la investigación y desarrollo de sistemas de visión artificial. Aunque en muchos casos el coste final de un dispositivo sea bajo, llegar a obtener un producto final comercial requiere de una inversión de recursos importante.

Tal es el caso de Google Inc., que recientemente está adquiriendo multitud de pequeñas compañías dedicadas a la robótica, entre las que algunas están dedicadas a la visión 3D, como es el caso de Industrial Perception Inc., que centra sus esfuerzos en el desarrollo de sistemas avanzados de visión artificial para entornos industriales.

Otro proyecto de Google más orientado a usuarios domésticos, es Project Tango, una tableta con capacidades de visión 3D cuyo sistema de visión

artificial está desarrollado por Mantis Visión [14], consistente en una integración de un sistema estereoscópico y de luz estructurada, que además se combina con el uso de acelerómetros y GPS para aumentar la precisión del mismo. Esto facilita, por ejemplo, el mapeado y escaneado en tres dimensiones de manera cómoda, la segmentación de objetos en el entorno, etc.

Para terminar con Google, uno de los proyectos robóticos más comentados en la actualidad es el de sus coches autónomos [5], capaces de manejarse por entornos reales complicados (aunque aún está en pruebas) mediante sensores y sistemas de visión en tres dimensiones. Entre estos sensores hay varios radares, GPS, cámaras 2D, etc. pero el corazón de todos ellos es un LIDAR con detección de 360º en horizontal y 26.8º en vertical con el que se genera un mapa completo y detallado del entorno como se puede ver en la *figura 2.2*.

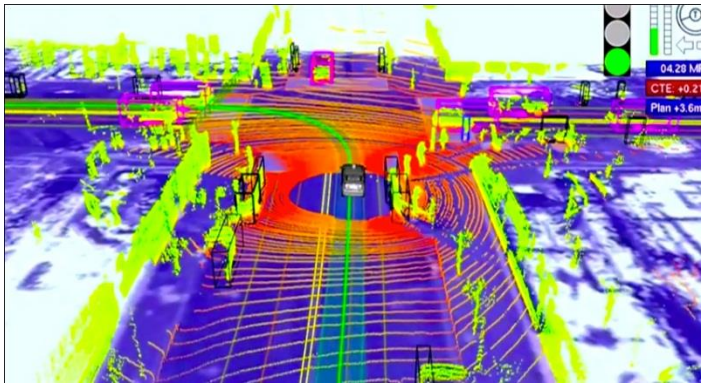

Figura 2.2. Google Car y mapa 3D generado [5].

Otra de las grandes empresas que ha entrado recientemente en el campo de la visión artificial es Microsoft, que desarrolla y comercializa el dispositivo Kinect con un hardware creado por la compañía PrimeSense. Kinect combina técnicas de visión estereoscópica con un mecanismo de visión estructurada para generar un mapa del entorno, con posibilidad de detección de personas y objetos de forma bastante precisa. Kinect fue desarrollado en primer lugar como dispositivo lúdico para la interacción con videojuegos, pero sus amplias posibilidades lo han llevado al terreno de la investigación como una forma de conseguir una fiable visión artificial 3D a un bajo precio.

En cuanto a los organismos de tipo público que más avances realizan en proyectos de robótica y visión artificial, los organismos que más desarrollos realizan son DARPA (Defense Advanced Research Projects Agency) y la NASA (National Aeronautics and Space Administration) [18].

La primera, que enfoca sus proyectos al ámbito militar, ha presentado recientemente sistemas de visión sintética que combinan la visión en 3D del entorno frontal con un sistema de proyección para facilitar a soldados información relevante sin obstaculizar la visión regular, de forma parecida a como funciona la realidad aumentada [4].

Por otro lado, la NASA posee un laboratorio de visión 3D para desarrollar sistemas para sus Rover. Uno de los prototipos más novedosos es un escáner 3D integrado en un brazo robot para obtener modelos completos de un determinado objeto.

Otro frente importante en el desarrollo de sistemas de visión artificial son los grupos de investigación en visión artificial y robótica de universidades y departamentos de I+D+i de pequeñas empresas, que contribuyen constantemente a la creación de nuevos algoritmos y métodos para el aprovechamiento de las tecnologías, en ocasiones con importantes avances en algunos apartados muy específicos.

Entre los desarrollos más punteros orientados a la visión artificial 3D están el reconocimiento de objetos (*figura 2.3*) mediante el uso de bases de datos que clasifican y almacenan imágenes de los mismos [12] o el reconocimiento y análisis de escenas concretas, ambos desarrollos de uno de los laboratorios de visión artificial del MIT [28].

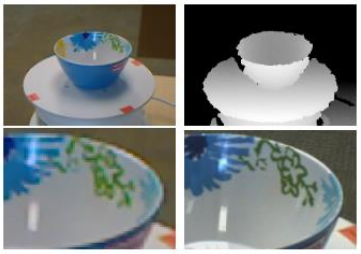

Figura 2.3. Reconocimiento de objetos [12].

Sin embargo, sin salir de territorio español podemos encontrar desarrollos relacionados, por ejemplo en el CAR (Centro de Automática y Robótica), que trabaja en un avanzado sistema de visión artificial estereoscópica para el manejo de robots teleoperados.

2.3 Robótica social y robótica de servicios.

Si bien en el presente proyecto la forma de interaccionar del robot con su entorno es mediante visión artificial, no debemos olvidar el propósito final,

dotarlo de un sistema lo más natural posible con objeto de hacer natural su situación en un ambiente real con personas.

Esto, que se conoce como robótica social, es uno de los aspectos que más relevancia tiene y tendrá en el futuro más inmediato, a medida que se pretenda introducir robots en situaciones reales con personas. Es una tarea compleja que no sólo requiere un reconocimiento del entorno adecuado, sino también una forma de interacción con el mismo que resulte natural, lo cual involucra diferentes aspectos, como es la comunicación con los agentes exteriores, la apariencia común en el contexto, etc.

Muchas empresas de todo el mundo llevan años intentando desarrollar robots antropomorfos lo más reales posibles, pero son las compañías japonesas las que más avances han realizado en este campo en las últimas décadas.

Un ejemplo de ello es el famoso robot ASIMO, de Honda [23], en la *figura 2.4*, y cuyo objetivo es ayudar a personas de movilidad reducida. Este ayudante con cuerpo de niño, posee avanzados sistemas de movilidad que, además de servir para su función objetivo, le dotan de una naturalidad de movimiento pocas veces vista en estos robots. Aunque tiene más de 10 años de vida, su última iteración, de 2014, ha incluido un sistema de visión estereoscópico capaz de distinguir personas adaptándolo así a diferentes usuarios.

Figura 2.4. ASIMO.

La también japonesa Toshiba ha presentado recientemente un robot humanoide, fidelizando al máximo posible los rasgos del mismo a los de una mujer, que además es capaz de comunicarse en lenguaje de signos. El objetivo de Aiko [22], que es como se llama esta mujer robot, es también la asistencia a personas. Con numerosos sensores y elementos móviles en todo

el cuerpo, es el robot con más similitud a un ser humano que se tiene constancia, a pesar de que no llegara al mercado hasta el 2020.

Otro famoso robot humanoide, creado en territorio español es REEM, de la empresa PAL Robotics [20]. Con varias iteraciones y una apariencia cada vez más humana, ha servido de ayuda en todo tipo de entornos, especialmente en congresos, centros comerciales y otros muchos lugares públicos.

En ocasiones no es tan importante la naturalidad de la apariencia y comportamiento del robot como la utilidad del mismo, sin descuidar por supuesto otros aspectos. Esto ha supuesto el desarrollo de robots que tienen un aspecto menos humano, pero que resultan igual de naturales en la sociedad actual altamente tecnificada y que además proveen servicios especiales según el entorno en el que se encuentran sin el elevadísimo coste asociado a la realización de formas e interacciones más reales.

Un ejemplo de esto se puede ver en robots guía en museos y atracciones turísticas, más simples que los humanoides mencionados pero que proveen el servicio al que están destinados de forma adecuada. Si bien hay muchos robots ejemplos de esto, algunos rasgos comunes en ellos son: una plataforma móvil para desplazarse, una pantalla para interactuar y consultar servicios, y algo parecido a una cara para dar una apariencia más amigable. Sin ir más lejos, estos son algunos de los elementos de Tito, el robot operativo en el museo de la ciencia de Valladolid desarrollado por el centro tecnológico Cartif y por tanto, con muchas similitudes con el robot objetivo del proyecto

Por último, y en relación con el robot que hará uso del sistema desarrollado en el presente proyecto, otro posible uso hoy día es la labor de botones en un hotel, donde puede proveer varios servicios como la información e interacción con los huéspedes. Aparte del robot que nos ocupa aquí, que se mencionará posteriormente, ha habido otros desarrollos recientes en el ámbito, como el A.L.O. Botlr [3] que se puede ver en la *figura 2.5*, un robot asistente en hoteles (en este caso de la compañía Aloft) capaz de interactuar con su entorno, con los huéspedes mediante una tableta, e incluso capaz de transportar objetos de pequeño peso hasta las habitaciones.

Figura 2.5. A.L.O. Botlr, robot asistente en hoteles.

Estas son sólo algunas muestras del presente de la robótica social que poco a poco está haciendo más natural y asequible el uso de robots en entornos y para propósitos sociales.

Capítulo 3:

Análisis de requerimientos

A continuación, entrando ya en el cuerpo principal del proyecto, se encuentra un análisis preliminar de la situación, los requerimientos del sistema que se pretenden alcanzar y la elección de un sistema de visión artificial adecuado para ello, aunque sin entrar en demasiados detalles de ello, pues se desarrollará en capítulos posteriores.

3.1 Análisis preliminar.

Antes de delimitar las funciones concretas, es importante conocer la situación y posibilidades de desarrollo de aplicaciones de visión y las funciones que éstas pueden realizar.

3.1.1 Análisis del robot y su emplazamiento.

El primer punto a considerar, y uno de los más importantes a la hora de desarrollar aplicaciones para un robot, es tener en cuenta el entorno en el que se va a desenvolver el mismo.

En este caso, el emplazamiento final de un robot que lleve este tipo de sistema es un lugar público con una afluencia media de personas. Se trata de espacios de grandes dimensiones y generalmente facilitando la movilidad de personas y grandes objetos, sin estrechos pasillos u obstáculos interpuestos que incomoden el movimiento, por lo que las aplicaciones deben estar adaptadas a funcionar en un rango de distancias relativamente amplio. Esto debe hacerse no sólo con el sistema de visión, sino con otro tipo de sensores que incorpore el robot final, especialmente los relacionados con la localización y el movimiento, como pueden ser láseres o sensores por contacto. Como ya se ha dicho, el primer robot al que irá destinada la visión artificial desarrollada es un robot de asistencia en hoteles, por lo que se adecúa perfectamente a lo anteriormente descrito, con un entorno y localizaciones amplias, como se observa en la *figura 3.1*.

Figura 3.1. Entorno ejemplo para el robot.

Sin embargo, otro dato a tener en cuenta es el uso del robot, y como principalmente este será un robot de interacción social, es relevante la manera en la que esta interacción se realice. La mayoría de robots de tipo social existentes, como el ejemplo de pruebas que se utilizará aquí, incorporan una pantalla táctil para interactuar con los diferentes menús, por lo que está previsto que la interacción con el robot se realice en un rango de distancias bajo o medio-bajo.

Unido a esto, el tipo de usuario que encontrará el robot en su recorrido es determinante a la hora de programar sus funciones. No es lo mismo saber que el robot estará entre personas con conocimientos de robótica y/o del funcionamiento interno del robot, que con personas ajenas a esto y que de entrada no tienen ningún conocimiento de cómo hacer funcionar el robot. En nuestro caso, los dos tipos de usuarios son posibles, y dependiendo de la función objetivo, ésta deberá adaptarse o no a esto.

Por último, igual de importante que el ambiente en el que se usará el robot son las características físicas del mismo. Como ya se ha dicho, se espera la utilización en un robot de interacción social, y es esto principalmente lo que

se potencia en su funcionamiento. Así, quedan relegadas a un segundo plano o desaparecidas algunas funciones comunes en robots, como la capacidad prensil, eliminada completamente debido a que carece de manos capaces de cerrarse y asir o similares, la capacidad de transportar objetos o funciones que no tienen que ver estrictamente con la interacción hombre-máquina.

3.1.2 Análisis de las funciones del robot

Por lo general, la visión artificial es sumamente importante en robótica, y no sólo en aplicaciones que pueden realizarse mediante ella, sino como complemento a otro sistema.

Así, se han considerado diversas posibilidades de actuación en el presente proyecto teniendo en cuenta los detalles extraídos del análisis preliminar y priorizando algunas características sobre otras. Algunas de las funciones que se han tenido en cuenta para su realización son las siguientes:

- **Interacción con un usuario común mediante gestos:** de tal forma que cualquier persona pueda solicitar al robot determinadas acciones o darle indicaciones. Por lo general, la solicitud de estas acciones se realiza con la pantalla táctil, pero está abierto a la posible solicitud mediante movimientos directamente delante de la pantalla u órdenes mediante gestos concretos.

- **Interacción con un usuario especial.** A diferencia de los anteriores usuarios, mediante algún reconocimiento único (gesto específico, un código QR o placa especial, etc.), el robot sería capaz de entrar en un modo especial en el que realizara acciones específicas o tuviera un control diferente para ciertos usuarios, preferentemente las personas encargadas del manejo o mantenimiento del mismo. Algunas de las posibles acciones en este modo serían el acceso a funciones especiales o el control más profundo del robot.

- **Seguir al usuario:** manteniéndole en todo momento dentro del campo de visión del robot y evitando confusiones con otras personas que pasen por el entorno, de forma que el robot siga al usuario correcto. Esto puede servir tanto como método de traslado del robot de manera fácil y precisa, como para mover el robot a petición de un usuario común para ver de lo que es capaz el mismo.

- **Distinguir las expresiones, la ropa y complementos u otras características más específicas de la persona:** pudiendo cambiar su actitud o los servicios a mostrar según las características de la persona (distinguiendo

emociones, abrigos, ropa de playa, cantidad y tamaño de objetos que puedan portar…).

- **Control del número de personas:** por ejemplo, para dar un aviso o alarma cuando haya gente esperando para ser atendida o cuando la cola para un servicio sea larga.

- **Ocio:** una de los objetivos del robot, además de proporcionar servicios relacionados con el entorno en el que se encuentra (en el descrito servicios de turismo o del hotel), consiste en el entretenimiento de las personas que interactúen con él, para lo cual se puede usar la visión artificial para aplicaciones de realidad aumentada (*figura 3.2*), juegos manejables mediante el movimiento, o el control de un avatar virtual.

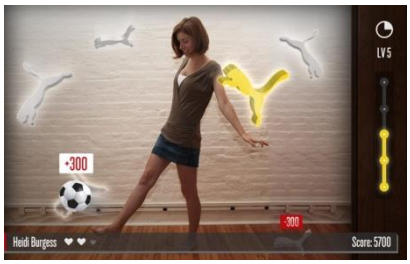

Figura 3.2. Juego de realidad aumentada.

- **Navegación y movimiento:** una de las aplicaciones más importantes de la visión artificial en robots móviles es la ayuda a la navegación y al propio movimiento del robot de diversas maneras. Algunas de las características relacionadas con ello que se podrían implementar son el SLAM, o mapeado del entorno a la vez que el robot se desplaza por el mismo o la detección de colisiones en el rango que puede abarcar la propia cámara.

- **Reconocimiento de objetos:** puede ser interesante dotar al robot de reconocimiento visual de objetos para distintas aplicaciones, como por ejemplo, aviso al personal de labores de limpieza o mantenimiento, levantamiento para los robots que incluyan extremidades prensiles, etc.

De entre los distintos sistemas anteriormente expuestos, algunos son especialmente interesantes y otros directamente desechables por no tener cabida en un robot de las características especificadas.

En primer lugar, la interacción mediante gestos de cualquier tipo de usuario es muy interesante y realizable con las tecnologías propuestas en este proyecto. Así, se podría controlar el movimiento del robot o su comportamiento según los gestos de las personas, determinando previamente un usuario que le comande.

Unido a esto, es interesante la posibilidad de que el robot siga al usuario según algún gesto concreto por requerimiento de los usuarios o para que el personal encargado traslade al robot a un punto deseado específico por algún propósito.

Las aplicaciones de ocio o reconocimiento de las características específicas de las personas son algo más complejas y por sí mismas cada una de ellas sería objeto de un estudio y desarrollo dedicado, por lo que no se abordarán en este proyecto. Sin embargo, se podrían incorporar en un futuro, sobre la base del presente proyecto, sin necesidad de recurrir a sistemas de visión adicionales.

Respecto al SLAM, el propio robot objeto del proyecto, y la mayoría de los robots de este tipo cuentan con un láser de largo alcance en dos dimensiones que provee esta función, y aunque un dispositivo de visión artificial de bajo coste lo podría realizar en tres dimensiones, la precisión y alcance serían mucho menores, pero a un coste demasiado elevado. Sin embargo, la detección de colisiones sí que es un objetivo interesante de plantear, que complementaría a los sensores ya dispuestos a tal efecto y vencería, de manera más o menos acertada, la deficiencia de que el láser sólo funcione en una determinada altura. No obstante, de nuevo esto podría ser objeto de estudio de un trabajo aparte tan extenso como este mismo.

Por último, aunque el reconocimiento de objetos podría resultar interesante, el hecho de que no todos los robots sociales, incluyendo al robot que se pretende utilizar, dispongan de un dispositivo para portar o levantar estos objetos hace que sea innecesario desarrollarlo.

Así, los objetivos que se van a fijar finalmente para este proyecto son los siguientes:

- Reconocimiento de personas y su forma física.
- Reconocimiento de gestos o posición del cuerpo.
- Seguimiento del robot a un usuario determinado y modificación del movimiento del robot de acuerdo a los gestos de dicho usuario.

3.2 Sistemas de Visión Artificial

Para llevar a cabo los objetivos planteados anteriormente es necesario un adecuado dispositivo de visión artificial que sea a la vez lo suficientemente potente y preciso, fácil de utilizar y de bajo coste.

En el mercado existe una amplia gama de sensores, cámaras y otros sistemas que pueden proveer visión artificial de diferentes maneras, siendo

éstos compatibles con diferentes sistemas y en un rango de precios también muy amplio. Algunos de ellos son:

- **Kinect,** en la *figura 3.3*: desarrollada por Primesense y comercializada por Microsoft, fue uno de los primeros dispositivos que proporcionaban visión artificial en tres dimensiones a un coste asequible para los usuarios comunes. Con características suficientemente buenas para muchas aplicaciones profesionales [11], es uno de los dispositivos que más ha hecho por llevar la visión artificial 3D a todo tipo de dispositivos, robots incluidos. Tiene una segunda iteración con mejores características, pero también un precio superior, que aun así es muy bajo en comparación con otros sensores de este tipo.

Figura 3.3. Kinect.

- **Xtion PRO [31]:** de los mismos creadores de Kinect y comercializado por Asus, mejora ligeramente las características de la cámara Kinect original a un precio algo superior, pero en prácticamente todo es igual a este, como se puede ver en la *figura 3.4*. Está orientado a su uso en el mercado de los usuarios de ordenador como alternativa a Kinect.

Figura 3.4. Xtion PRO [31].

- **Leap Motion** (en la *figura 3.5*) **[13]**: utiliza métodos de luz estructurada, es decir, LEDs infrarrojos, para captar lo que pasa a su alrededor, especialmente los gestos, para lo que está diseñado. Sin embargo, sus diferencias con Kinect y otros dispositivos de visión 3D son claras: tiene un mayor ángulo de acción (120º-150º), pero un alcance mucho menor, sólo

hasta 60 cm. Esto lo hace especialmente útil para su uso con ordenadores portátiles y otros dispositivos de interacción muy cercana.

Figura 3.5. Leap Motion [13].

- **Riegl [21]:** es una gama de láseres capaces de actuar en tres dimensiones, pero especialmente caros y que se usan sólo en aplicaciones militares o de muy alto coste y precisión. Además, tal y como se ve en la *figura 3.6*, tiene un tamaño considerable, por lo que no es un dispositivo adecuado para según qué aplicaciones robóticas.

Figura 3.6. Láser Riegl [21].

De entre los dispositivos presentados o la posibilidad de fabricación de forma más rudimentaria de una cámara estereoscópica con dos cámaras más simples se ha elegido el sensor Kinect por varios motivos.

En primer lugar, es un sensor que ya se encontraba disponible para su uso previo, pero que aunque así no fuera, tiene un coste bajo para lo que ofrece, haciéndolo tremendamente atractivo y cumpliendo así el objetivo de minimizar el coste del proyecto final en su totalidad.

Además, sus características cumplen en gran medida con las exigencias necesarias para el desarrollo de las funciones pretendidas, algo que alguno de los otros dispositivos, como Leap Motion, no podría hacer, al centrarse en un rango muchísimo menor, a pesar de ser más preciso.

Por otro lado, de entre todos los dispositivos que se podrían utilizar, es el que tiene un mayor ecosistema de aplicaciones para su uso y usuarios trabajando en él, especialmente en ROS, donde se pretende realizar la programación. Existen drivers y programas especialmente diseñados para su uso, por lo que se pueden aprovechar muchas cosas ya desarrolladas con anterioridad, y lo más importante, el uso del mismo es directo, sin pasar por configuraciones previas como sería el caso de otros dispositivos o cámaras estereoscópicas fabricadas de forma casera.

También cabe tener en cuenta qué modelo de Kinect utilizar, puesto que existen tres distintos en la actualidad. El último modelo, puesto a la venta en septiembre de 2014 se ha desechado en primer lugar debido a la falta de soporte para su uso, sobre todo en sistemas de tipo abierto como son Linux o ROS. Después, si tenemos que elegir entre las dos Kinect de primera generación (Kinect for XBOX y Kinect for Windows), se puede considerar indistinta la elección, puesto que no hay diferencias especialmente apreciables entre ambas. La segunda tiene un modo de visión cercana, preparado para detectar mejor al usuario en un rango más corto de acción, pero en la práctica este modo no se utilizaría, por lo que la elección ha seguido indiferente. Finalmente, se ha optado por Kinect para Xbox, al ser la que se encontraba disponible desde el inicio del proyecto y con la que se ha trabajado a lo largo de todo el desarrollo del mismo.

3.3 Kinect como sistema de VA.

Una vez elegida la cámara Kinect como sistema de visión artificial, es necesario analizar al detalle algunas de sus características y conocer su funcionamiento para así poder planificar la realización de las funciones objetivo.

Kinect es un dispositivo innovador en cuanto a funcionamiento, ya que combina las dos tecnologías más utilizadas en la actualidad para conseguir visión en tres dimensiones: la visión estereoscópica y la luz estructurada. Son dos métodos complementarios, cuya combinación mejora enormemente su efectividad por separado.

Concretamente, como explica Alhwarin [1] y se ve en la *figura 3.7*, la toma de imágenes se realiza mediante la combinación de las dos cámaras RGB como se haría mediante un clásico método estereoscópico, explicado previamente en el apartado 2.2. Sin embargo, a esta imagen se le añaden los datos del sensor infrarrojos (emisor y receptor de estos puntos de luz) para

obtener la distancia precisa de cada punto en el espacio, lo que ayuda a calibrar aún mejor las distancias, obteniendo así un conjunto de datos mucho más fiables.

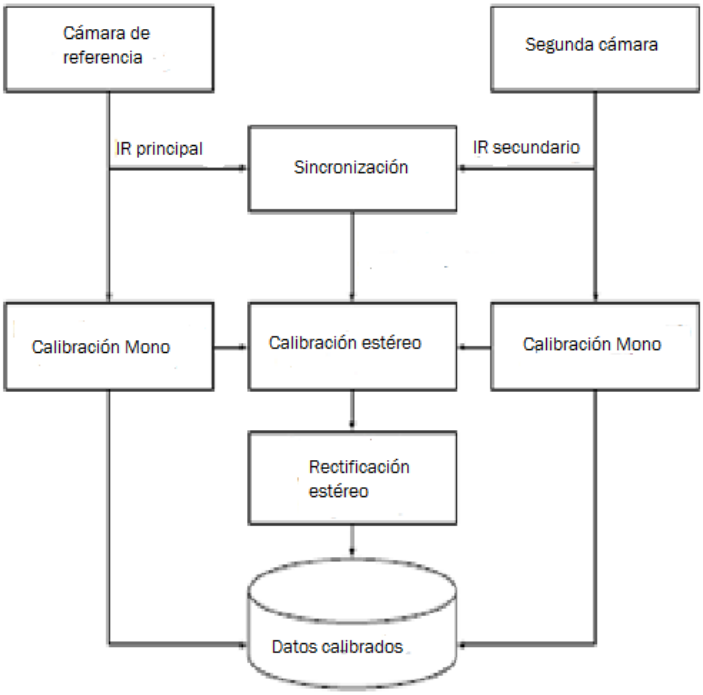

Figura 3.7. Diagrama de funcionamiento de la calibración de Kinect [1].

Las especificaciones detalladas de Kinect se pueden ver en el anexo correspondiente, pero algunas especialmente interesantes para tener en cuenta son:

- **Resolución del sensor IR:** 1280x1024 píxeles a 15 fps o 640x480 píxeles a 30 fps. Las distintas resoluciones son consecuencia del alto coste computacional requerido para el tratamiento de tantos puntos. La menor resolución es suficiente para la mayoría de aplicaciones y facilita la detección de un gran número de puntos (figura 3.8), pero el tratamiento de tal cantidad aún así puede traer ralentizaciones innecesarias, por lo que puede ser interesante reducir aún más esta resolución para según qué casos.

Figura 3.8. Puntos de luz proyectados por el sensor IR.

- **Rango de operación:** 0,4m – 3,5 m. Por un lado, el límite inferior hace imposible utilizar el dispositivo demasiado cerca, pero esto no resulta problemático en nuestro caso dado que no hay ningún problema puesto que las interacciones que se esperan con él se deben dar a una determinada distancia. El límite superior hace imposible, por ejemplo, el uso de SLAM en un entorno tan amplio como el que se pretende utilizar, pero de nuevo, no es el caso de las funciones aquí desarrolladas y únicamente habrá que tener especiales consideraciones en el seguimiento del robot a la persona, que no se debe hacer a demasiada distancia.

- **Campo de visión del sensor IR:** 57º en horizontal, 43,5º en vertical. Es uno de los mayores inconvenientes del sistema que han sido observados, en especial el campo vertical, del que se hablará en el apartado siguiente, porque dicho ángulo limita a su vez la distancia de interactuación al no poder englobar a una persona completamente. Sin embargo, es subsanado en parte con un movimiento en vertical del propio dispositivo.

- **Rango de inclinación:** 54º (27º hacia arriba y otros 27º hacia abajo), que sirven para modificar la posición del campo de visión en el caso de que la altura a la que deba ser colocado el dispositivo no sea la adecuada para los propósitos, facilitando cierta versatilidad por parte del sistema en la captación de imágenes.

Las anteriores características son las que hacen que la cámara funcione como lo hace, en especial la nube de puntos que genera. Una infografía de algunas de las anteriores especificaciones se puede ver en la figura 3.9. Esta nube, cuyo propósito y cometido se explicará brevemente en el capítulo posterior, basa algunas de sus características en las anteriores especificaciones.

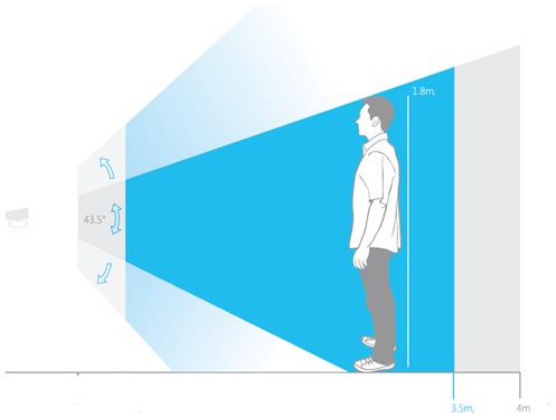

Figura 3.9. Rangos de visión en Kinect.

Capítulo 4:

Programación basada en componentes. ROS

Los desarrollos actuales en robótica explicados en el capítulo anterior se deben apoyar en un software y unos programas implementados en él para funcionar correctamente, no toda la innovación viene por parte de los sistemas hardware. Aquí es donde entran diferentes arquitecturas, sistemas operativos o sistemas de desarrollo, cada uno con ventajas e inconvenientes pero muy distintos entre sí, aunque el sistema por excelencia es ROS [24], que cuenta con una comunidad de desarrolladores enorme.

4.1 Introducción. Sistemas operativos para robots.

Hasta hace unos pocos años, los desarrollos en robótica, más propios de grandes empresas, estaban ligados a entornos software propietarios y cerrados, que obligaban a aquel que quería trabajar con un robot de una marca a usar su propio software con licencia. Como máximo, al margen de esto, se lograban crear en algunas importantes universidades pequeños conjuntos de marcos y herramientas para facilitar la programación de algunos robots, pero nunca nada de forma masiva.

Algunos de estos sistemas son:

- MOOS [16]: desarrollado por la universidad de Oxford como plataforma para el desarrollo de robótica móvil con una estructuración en capas y programado en C++.

- Orca [19]: utilizado en universidades australianas, se basa en la programación de módulos que se incorporan al robot.

- YARP [32]: una colección de programas intercomunicados entre sí desarrollada por el MIT para su uso en robótica humanoide.

- Urbi [29]: basado en componentes o módulos y desarrollado en Francia, es uno de los más recientes.

Sin embargo, todos ellos han quedado desplazados, al menos en los desarrollos robóticos fuera de grandes empresas y corporaciones, por ROS, que son las siglas de Robot Operating System (Sistema Operativo Robot).

Creado en 2007 en el laboratorio de inteligencia artificial de Stanford y desarrollado por Willow Garage en sus primeros años, se basa en la experiencia con otros sistemas robóticos para proveer un conjunto de entornos, herramientas y control de elementos en robótica. En definitiva, proporciona servicios propios de un sistema operativo en robots.

Como se ha podido ver, una característica común en varios de los sistemas robóticos es la programación basada en componentes o programación modular, en la que se basa precisamente ROS. Esto significa que existen distintos módulos que proporcionan distintas funciones, y el conjunto de todos ellos hace actuar al robot como se desea, no siendo ninguna de ellas indispensable para el funcionamiento como tal, sino para alguna de sus partes.

Suponiendo un robot semejante al de este propio proyecto, tendríamos un módulo para el control de los motores y el movimiento, otro módulo para la gestión de la información recibida por los sensores, otro para la visión artificial, etc.

Así, el resultado final del funcionamiento del robot es la suma de las funciones de cada uno de los módulos, como se resume en la *figura 4.1*, pero con la posibilidad de eliminar o incorporar nuevos módulos en caso de quitar o añadir alguna función deseada o elemento nuevo.

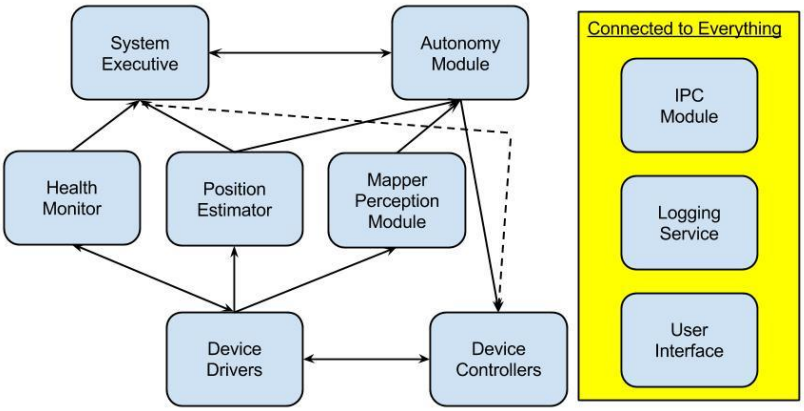

Figura 4.1. Esquema ejemplo de módulos en ROS.

Pero otra de las grandes ventajas de ROS ha sido su expansión, al tratarse de una plataforma abierta, que ha facilitado su uso en diferentes desarrollos e implantación en todo tipo de robots. Además, la comunidad desarrolladora comparte gran parte de los programas hechos y esto posibilita la reutilización de muchos de ellos con pequeñas adaptaciones, para no tener que desarrollar desde cero en cada robot funciones iguales a las que ya realiza otro robot.

4.2 Conceptos básicos de ROS.

Es importante manejar algunos conceptos básicos en ROS antes de programar aplicaciones en él, e incluso antes de diseñarlas, puesto que la propia arquitectura distribuida hace diferente la estructuración del código y las posibilidades del robot.

4.2.1 Modelo editor-suscriptor y paso de mensajes.

La base más importante de ROS es la programación en módulos, la cual se apoya a su vez en un modelo editor-suscriptor, clásico entre los sistemas de intercambio de información. Es un modelo de comunicación que se diferencia del resto de su clase en que es asimétrico y en que los productores pueden enviar información al consumidor sin que éste tenga que pedirla. Normalmente en robótica, las diferentes partes del robot no pueden actuar

por su cuenta, los actuadores deben apoyarse en la información de diferentes sensores y viceversa, por lo que se hace importante el paso de mensajes entre los diferentes módulos para que los distintos sistemas estén comunicados entre sí y el conjunto funcione adecuadamente.

Para ayudar a comprender este paso de mensajes, algunos conceptos básicos al manejar ROS son:

- Nodos: son las unidades individuales del sistema operativo, ejecutables que realizan una determinada función o tarea según estén programados y que se comunican con otros nodos.

- Topics: mecanismos de comunicación entre nodos. Éstos pueden publicar mensajes en los topics o se pueden suscribir a ellos para leer los mensajes de otros nodos.

- Mensajes: una estructura de datos que puede ser de muchos tipos y que contiene información que los nodos desean pasarse entre sí a través de los topics.

Para ayudar a que todos los elementos anteriores funcionen, existe un maestro, que se inicializa mediante el comando:

```
roscore
```

y que asegura que las comunicaciones estén correctamente establecidas y los nodos se encuentren entre sí. Es el único elemento indispensable para hacer funcionar ROS.

El SO incorpora también un modelo de cliente-servidor que puede ser útil en determinadas situaciones y se puede implementar con facilidad; pero no es indispensable para el sistema y de hecho en el desarrollo de este proyecto no ha sido necesario utilizarlo, como se verá en capítulos sucesivos.

4.2.2 Versiones, instalación y robots soportados.

Debido a la amplia difusión que ha tenido ROS, en poco tiempo se ha desarrollado rápidamente y cuenta con múltiples versiones cada una con nuevas funciones y aspectos mejorados sobre la anterior. Sin embargo, esto a veces es un problema en el desarrollo de aplicaciones, debido a la continua adaptación que hay que realizar en los programas para ajustarse a la versión actual.

Aunque en el momento de realización de este trabajo va por su novena versión, ROS Indigo, el robot objeto de pruebas cuenta con la séptima, ROS Groovy, por lo que es con la que se trabajará en adelante.

Al no ser un sistema operativo completo, sino una colección de librarías, herramientas y reglas para simular un sistema operativo en robots, ROS necesita ser instalado sobre un sistema operativo completo, como es el caso de Windows, Mac OS o distribuciones libres basadas en Linux.

Debido a que el propio ROS está estructurado y hecho como un sistema libre, es Linux, y concretamente su distribución Ubuntu, la que más adecuada y preparada resulta para su instalación. Específicamente, la versión Groovy de ROS se puede instalar en las versiones de Ubuntu 11.10, 12.04 y 12.10, pero en este proyecto se ha usado la versión 12.04.

Los comandos previos para configurar los repositorios y llaves para la correcta instalación como recomiendan los propios creadores del sistema operativo son:

```
sudo sh -c 'echo "deb
http://packages.ros.org/ros/ubuntu precise main" >
/etc/apt/sources.list.d/ros-latest.list'
```

```
wget http://packages.ros.org/ros.key -O - | sudo apt-
key add –
```

```
sudo apt-get update
```

Y el comando para iniciar la instalación de todos los módulos para el correcto funcionamiento de ROS es:

```
sudo apt-get install ros-hydro-desktop-full
```

Con la posibilidad de sustituir los dos últimas palabras del comando por el nombre de algún paquete de ROS específico, en el caso de sólo querer instalar el mismo.

Por último, al igual que se ha destacado la gran cantidad de personas trabajando con ROS, hay que destacar la gran cantidad de robots que hacen o pueden hacer uso de esta plataforma, aunque no sea su principal. Éstos incluyen robots comerciales de todo tipo que existían previamente a la aparición de ROS, robots comerciales desarrollados expresamente con ROS, robots para el trabajo de investigación o simplemente robots realizados por aficionados que se aprovechan de ROS como una forma fácil de programarlo y controlarlo.

Entre estos robots se incluyen el PR2, creado por Willow Garage, es uno de los más veteranos incorporando ROS; Turtlebot, una plataforma móvil que utiliza como base un robot comercial, Roomba; el ya mencionado en el capítulo anterior Reem, de la española PAL Robotics; y muchos otros de tipos variados (ver *figura 4.2*) menos conocidos pero que son una prueba más de la versatilidad de este sistema en su implantación en robots.

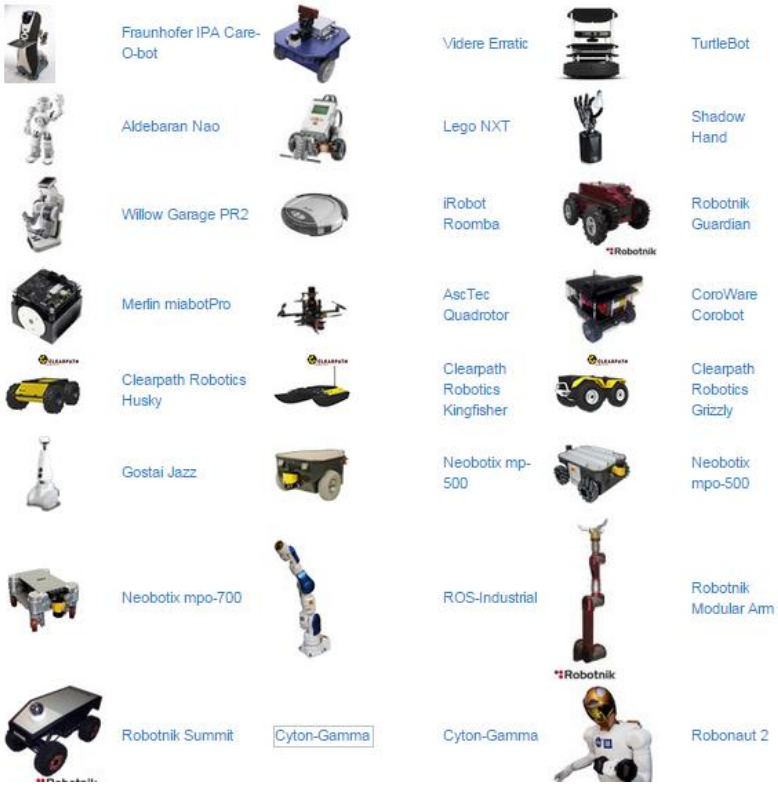

Figura 4.2. Ejemplos de robots que incorporan ROS [24].

4.3 Programación en ROS.

Aparte de los ya reseñados motivos del éxito de ROS como plataforma robótica, como son su arquitectura modular o su difusión al ser un sistema libre, otra causa de este éxito es la gran cantidad de ayudas y herramientas para el programador existentes a la hora de crear nodos y funciones para un robot. Desde conceptos específicos sólo aplicables en este ámbito hasta herramientas de depuración y visualización como las tienen otros sistemas de programación, todos ellos están enfocados a facilitar las tareas de desarrollo de sistemas robóticos.

4.3.1 Conceptos avanzados: TF y nubes de puntos.

Parte del éxito de ROS, además de las ventajas anteriormente descritas, es la integración de diferentes conceptos teóricos que ya existen en otros soportes de programación e implementación y que son significativamente importantes en robótica, como los sistemas de referencia y las nubes de puntos.

Transformed Frames

Cuando hablamos de robots de todo tipo, incluso el más simple de ellos, con apenas uno o dos grados de libertad, un concepto que nada tiene que ver con la programación, sino con la mecánica del mismo, es el de la transformación entre sistemas de referencia.

Un robot está compuesto por eslabones y articulaciones, y a cada eslabón le corresponde un sistema de referencia colocado y orientado según la movilidad de éste. De esta manera, mediante unas determinadas variables podemos parametrizar y conocer la posición exacta de cada uno de los elementos del robot.

ROS aplica este concepto y ofrece un conjunto de variables, herramientas y ayudas específicas para tratar con estas transformaciones (*transformed frames*) de forma que podemos conocer los valores de unos sistemas de referencia con respecto a los otros. Debido a estas dependencias, la mayoría de tf se publican como padres o hijos de otros en un árbol de sistemas de referencia.

Un ejemplo está en el robot de la *figura 4.3*. Todos sus elementos móviles o eslabones tienen un sistema de referencia y tf ayuda a conocer la distancia y orientación entre algunos de ellos (padres e hijos, puesto que normalmente algunos dependen directamente de otro), de forma que la posición exacta del robot en un determinado momento estará determinada por el conjunto de valores de sus tf en ese momento.

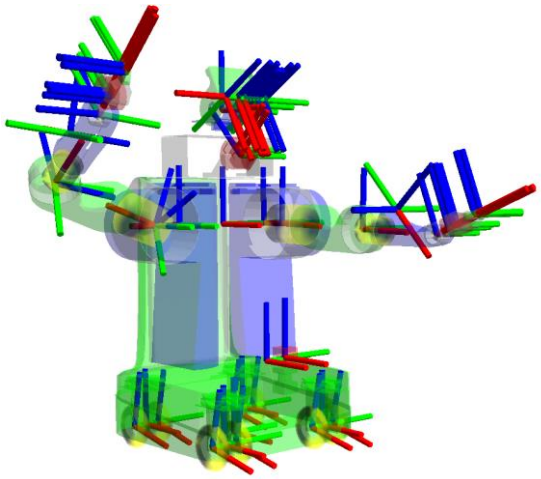

Figura 4.3. TF marcados sobre las articulaciones de un robot.

El concepto de tf se utiliza como cualquier otra variable de ROS, y su publicación se realiza mediante topics. Sin embargo, el topic en el que se publican, /tf, es especial, y no está sujeto a las normas de editor-suscriptor normales como el resto, sino que en la programación se utilizan funciones y clases especiales para obtener los valores deseados de los sistemas de referencia y transformaciones.

Nubes de puntos

Además de los *transformed frames*, otro concepto muy importante, esta vez relacionado directamente con la visión artificial en tres dimensiones es el de las nubes de puntos. Una nube de puntos, como su propio nombre indica, es un conjunto de puntos en el espacio de los que se puede conocer determinadas propiedades. En concreto, en visión artificial, las propiedades básicas que más interesantes pueden resultar son la posición xyz con una determinada referencia, la distancia de la referencia al punto en cuestión o el color de dicho punto. Sin embargo, otras propiedades más complejas que no dependen del punto en sí, sino de los puntos cercanos, podrían ser la normal o la curvatura, por poner un ejemplo.

Es posible conocer todas estas propiedades con los utensilios adecuados, como es Point Cloud Library (PCL), un conjunto de bibliotecas que contienen funciones, algoritmos y herramientas para el tratamiento de las nubes de puntos, como lo que se puede ver en la *figura 4.4*. Lo normal en un dispositivo que capta una nube de puntos es obtener la posición física de cada uno de estos puntos y nada más, por lo que para tratamientos más avanzados como la detección de superficies, el análisis de un conjunto de puntos, etc., es un instrumento excelente.

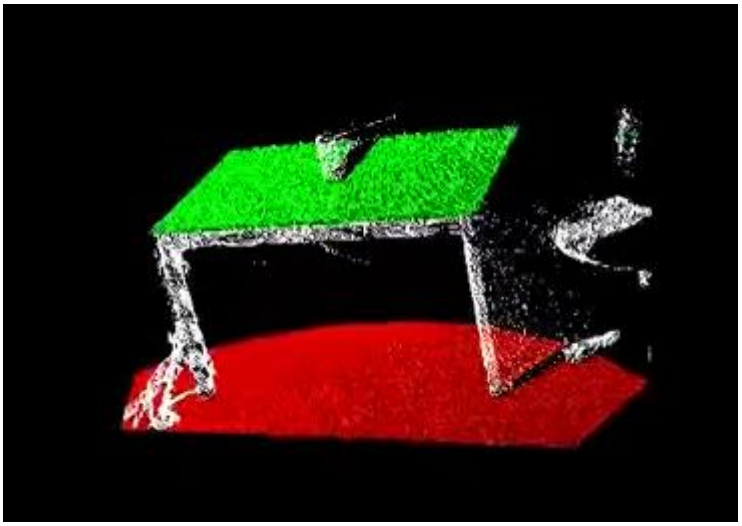

Figura 4.4. Tratamiento de una nube de puntos con PCL.

PCL se incorpora a ROS para ayudar con el tratamiento de nubes de puntos típicas de los sistemas de visión 3D, y es una ayuda esencial para tratar con la nube de puntos que entrega Kinect.

4.3.2 Creación de paquetes y ejecutables.

Como ya se ha explicado, ROS ofrece una arquitectura de funcionamiento basada en componentes independientes o módulos que hace que sea fácil combinar el trabajo propio con otros programas desarrollados por la comunidad que ROS tiene detrás.

En concreto, el funcionamiento de ROS en este aspecto es extremadamente simple. Se pueden crear nodos que ofrezcan una función o parte de ella a un robot, y estos nodos se agrupan en módulos o paquetes que se aplican al robot. Pero es necesario considerar algunos extremos, por ejemplo en el caso de crear un nodo o paquete desde cero, en el que hay que añadir a los archivos correspondientes (que aquí es el archivo CMakeLists), referencias a los demás paquetes y bibliotecas de funciones utilizados, para que el sistema los encuentre y trabaje correctamente. A esto se le llama dependencias de un paquete en ROS.

De igual manera, se pueden descargar paquetes creados por otras personas y, adaptándolos si es necesario, hacerles funcionar en el propio robot. De hecho, la instalación de ROS viene con un amplio conjunto de paquetes con muchas funcionalidades distintas, algunas incluso adaptadas a robots concretos.

Por otro lado, a la hora de implementar una funcionalidad en el robot es útil crear ejecutables o launchers, que lanzarán y pondrán en funcionamiento de una determinada manera los nodos programados. Así, con un mismo conjunto de nodos es posible implementar distintas funciones o características o adaptarlas a diferentes entornos, creando diferentes launchers que configuren el sistema y las funciones de una manera determinada.

Por último, todo lo anteriormente descrito se encuentra dentro de los espacios de trabajo o workspace, que es el lugar en el que se agrupan los nodos que van a trabajar juntos, presumiblemente en un mismo robot.

Existen explicaciones respecto a estos apartados y muchos otros en la wiki oficial de ROS [25].

4.3.3 Herramientas de depuración y visualización.

Para asistir al programador mientras desarrolla las aplicaciones y el código de los nodos del robot, existe un paquete de numerosas aplicaciones que ayudan a visualizar por pantalla de manera más o menos fácil valores o conceptos en tiempo real o posterior al funcionamiento.

Debido a la gran cantidad de herramientas para esto de que ROS dispone, sería demasiado extenso describir todas y cada una aquí. Por ello, se enumeran y describen brevemente a continuación aquellas utilizadas en el desarrollo de los programas objeto del presente trabajo.

- Rostopic: es un conjunto de comandos que sirve para obtener datos relacionados con los topics de ROS. Así, por ejemplo, el comando rostopic list lista los topics que se encuentran en el momento abiertos, o rostopic echo muestra por pantalla los mensajes publicados.

- Rqt_console: es una herramienta muy útil que muestra por pantalla mensajes de todo tipo, desde avisos de error cuando algún elemento no funciona como debería, hasta diferentes datos que el programador mismo ordene mostrar (ver *figura 4.5*). Es especialmente útil a la hora de ver en tiempo real o guardar para un análisis posterior valores de variables importantes en el código, normalmente con el fin de comprobar o depurar el funcionamiento del mismo.

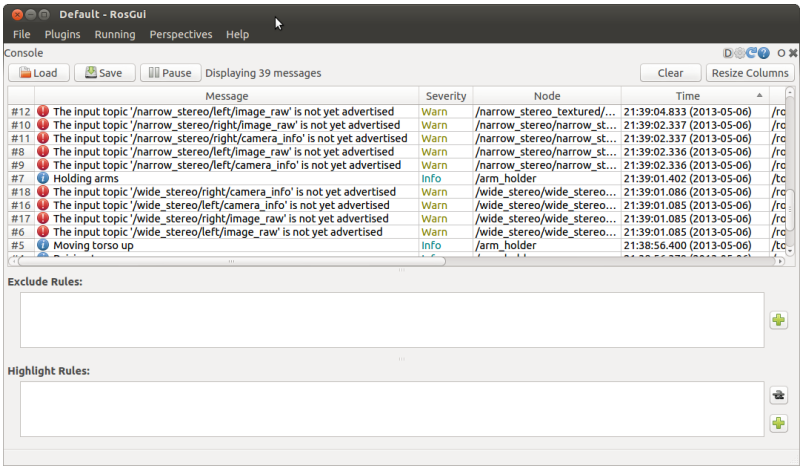

Figura 4.5. rqt_console.

- Rqt_graph: visible en la *figura 4.6*, es una herramienta de ROS utilizada para mostrar de manera visual el conjunto de nodos, topics, sus relaciones publisher/suscriber y valores que intercambian entre sí.

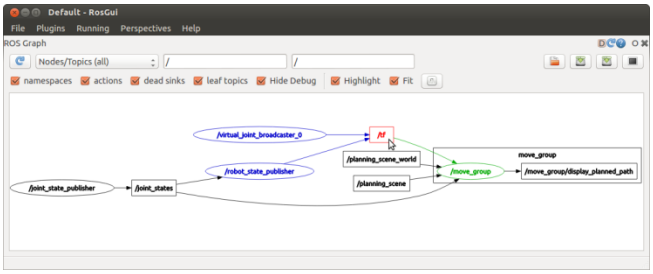

Figura 4.6. rqt_graph.

- Rviz: es un potente programa de visualización que permite representar información de muy diversos tipos de forma fácil y muy personalizable, como se observa en la *figura 4.7*. En el caso que nos ocupa, para visión artificial, puede representar imágenes obtenidas con una cámara o puede representar nubes de puntos para aplicaciones de visión 3D. Incluso puede representar estos datos junto al propio robot simulando el entorno real dentro del programa. Además, ofrece una configuración para representar también todos los tf involucrados en un determinado programa y las relaciones entre ellos. Sin embargo, uno de sus desventajas es el alto contenido en recursos que puede llegar a consumir con las formas más avanzadas de visualización, por lo que no conviene utilizarlo más que para depuración mientras se desarrolla el programa.

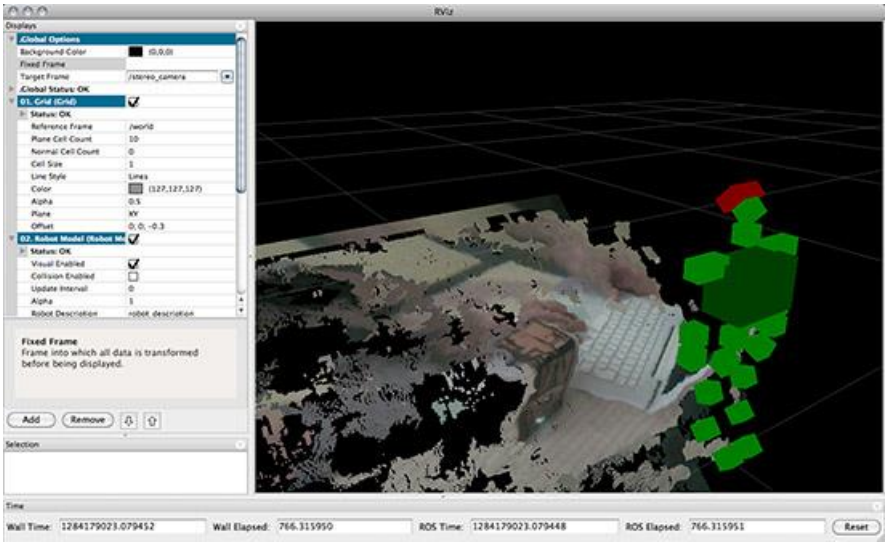

Figura 4.7. Programa de visualización Rviz.

- Turtle_tf: no es como tal una herramienta de ROS, sino un paquete tutorial que sirve para comprender el funcionamiento de los *transformed frames* de manera visual. Una vez se ejecuta, muestra por pantalla un espacio 2D en el que una o más tortugas dibujadas se mueven de acuerdo a la programación que se les haya dado. Si bien, como ya se ha dicho, no es una herramienta, ha sido especialmente útil para trabajar sin tener el robot presente, puesto que los mensajes de velocidad de la tortuga eran equivalentes a los del movimiento con el joystick del robot.

- Tf view_frames: es una herramienta muy útil para tratar y documentar adecuadamente los *transformed frames*, ya que genera un pdf con todos los activos en el momento, sus dependencias y algunos datos útiles de los mismos (ver *figura 4.8*).

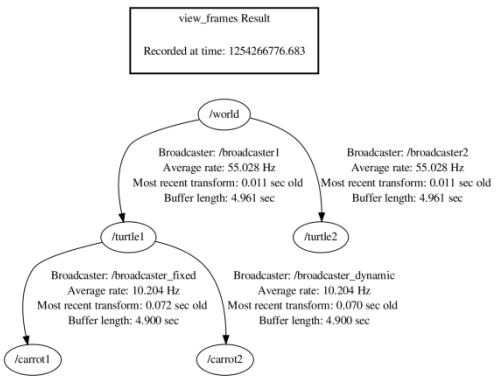

Figura 4.8. Ejemplo de pdf generado por view_frames

Capítulo 5:

Los robots Sacarino y Edubot

Aunque el objetivo del proyecto sea el diseño de una aplicación de visión artificial para un robot de tipo social genérico, es necesario conocer algunos ejemplos de este tipo de robots para poder concretar adecuadamente determinados aspectos y especificaciones de los sistemas desarrollados. Uno de los robots que implementarán estos sistemas será Sacarino 2, la segunda iteración de un robot botones diseñado y fabricado por Cartif para su uso y asistencia en hoteles.

Sin embargo, debido a que no está completamente desarrollado en el momento de realizar este proyecto, se han utilizado otros dos robots para las pruebas pertinentes con la interacción mediante visión artificial: uno de ellos es el robot plataforma móvil Edubot, que tiene fines educativos y su sistema de movimiento es muy parecido al usado por Sacarino 2 para moverse; el otro es Sacarino, la primera versión del robot objetivo y con el que comparte muchas características como se verá a continuación.

Aun así, se ha mantenido una estrecha comunicación con la persona encargada del diseño de Sacarino 2 para tener en cuenta todos los aspectos relacionados con el hardware que sean importantes de cara al presente proyecto, y también se han intentado alcanzar los objetivos del proyecto teniendo en cuenta los posibles usos de la aplicación de visión artificial en otros robots con posibles estructuras diferentes.

5.1 Introducción a los robots.

Sacarino es la primera versión del robot que se pretende haga uso de la tecnología de visión artificial desarrollada. Es un robot para la asistencia en hoteles con capacidad de desplazamiento e interacción con las personas de su entorno [26]

Consta de dos elementos diferenciados. Por un lado, una parte superior de aspecto humano, y más concretamente de niño, como se ve en la *figura 5.1*. Para ello, cuenta con una cara, brazos y otros elementos con capacidad de comunicar determinados estados, información y emociones simuladas, como los ojos y párpados o una matriz con leds en el lugar de la boca. Además de todo lo anterior, delante de lo que sería su pecho lleva incorporada una pantalla táctil para interactuar y controlar el propio robot adecuadamente.

Por otro lado, en la parte inferior consta de una base móvil vista en la *figura 5.2* que lo dota de movilidad completa en el entorno gracias a un avanzado sistema de movimiento de las ruedas, además de numerosos sensores para posicionarse y moverse correctamente, como por ejemplo ultrasonidos, un sistema láser, etc. Además, dispone de un cajón portaobjetos y un portamaletas para facilitar su uso como botones de hotel.

Figura 5.1. Parte superior de Sacarino

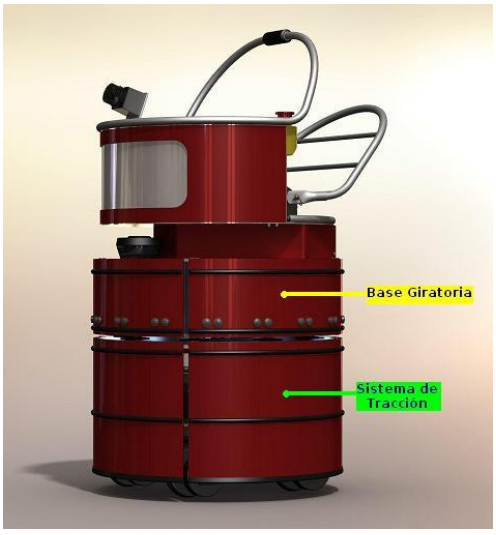

Figura 5.2. Parte inferior de Sacarino

En esta primera versión del robot, además de funcionar y servir para los objetivos propuestos, se ha recopilado mucha información de su uso en un entorno real con seres humanos, viendo así sus principales virtudes y las

cosas mejorables de cara a futuros diseños de este y otros robots semejantes.

Así es como, varios años después de este primer desarrollo, se ha decidido realizar una segunda versión mejorando la primera y adaptándolo a las nuevas necesidades del hotel, en base al *feedback* recibido anteriormente.

Sacarino 2 es superior a su predecesor en prácticamente todo, realizando mejor sus funciones y combinando lo que funcionaba adecuadamente del anterior modelo con nuevas especificaciones.

Para observar mejor los avances realizados, lo preferible es hacer algunas comparaciones entre las dos versiones y observar adecuadamente las diferencias entre los dos robots:

- En la cara las diferencias son mínimas, como se aprecia en la *figura 5.3*, pero por ejemplo se ha eliminado el micrófono de la frente del robot y los leds de la boca son sustituidos por una matriz parecida pero más versátil y sofisticada. También se modifican algunos mecanismos internos para el movimiento de parpados, pero no tienen demasiada relevancia.

Figura 5.3. Cara de Sacarino 2.

- Aunque la pantalla táctil se ha mantenido exactamente igual, y por tanto conserva plenamente su funcionalidad, los brazos en este caso han perdido la movilidad que tenían en la anterior versión. Además, aunque se detallará posteriormente, la cámara Kinect se espera que se coloque sobre la pantalla en el pecho del robot de manera bastante disimulada y en consonancia con el aspecto del robot, como se puede ver en la siguiente *figura, la 5.4*.

Figura 5.4. Kinect incorporado en el chasis del robot.

- La parte inferior del robot (en la *figura 5.5*) también se ha simplificado, transformando toda la parte inferior del anterior robot en unas piernas que no cumplen ninguna función aparte de la propia estética. No obstante, la parte más inferior, la plataforma móvil sobre la que se mueve el robot, se mantiene aunque modificando su comportamiento mecánico (posteriormente descrito en este mismo capítulo). También se mantiene el láser 2D con que contaba la primera versión.

Figura 5.5. Parte inferior anterior de Sacarino 2.

Sin embargo, todo lo anteriormente expuesto, aunque sí está pensado y diseñado adecuadamente, aún no se encuentra completamente desarrollado y construido, por lo que no es posible utilizar el sistema de visión artificial con el robot final ni con su plataforma móvil.

En su lugar, para realizar todo tipo de pruebas se hará uso del robot Edubot, un robot desarrollado también por Cartif como parte de un proyecto anterior [2], con fines educativos y de investigación como el que nos ocupa, y que sí está disponible durante la realización del proyecto. En esencia, este robot se trata de una plataforma móvil de un tipo semejante a la que posibilita la movilidad de la segunda versión de Sacarino (no así la primera), por lo que sirve perfectamente a los propósitos de usarla como si se estuviera controlando el propio robot asistente.

Además, más importante aún es el hecho de que el sistema de control y el sistema operativo (ROS) que controla los diferentes robots son los mismos,

por lo que las adaptaciones en los programas que se desarrollen aquí apenas necesitarán una adaptación al robot final, tan sólo ciertos ajustes en valores concretos.

5.2 Especificaciones físicas de los robots

Como ya se ha dicho, el robot Edubot es básicamente una plataforma móvil, y por tanto, centra casi toda su estructura en la mecánica y control de las ruedas. Sin embargo, cuenta también con otros elementos que le ayudan en esto, como son algunos sensores para evitar choques o caídas. A su vez, ambas versiones de Sacarino cuentan con una plataforma inferior semejante, aunque con algunas diferencias entre los tres robots.

Por todo esto, a continuación se van a explicar los aspectos y características que pueden ser de utilidad para el presente proyecto y su realización, y las diferencias principales entre los robots.

5.2.1 Plataforma mecánica móvil

Hablando en primer lugar de Edubot, la estructura mecánica del robot tiene forma de caja hexagonal, para poder incluir dentro los elementos de control importante y poder sostener encima otros elementos adicionales también y colocar por sus paredes algunos sensores de forma más distribuida. En el caso de los robots Sacarino, la plataforma móvil es circular, con una mejor estética general y problemas respecto al exterior, pero también con un peor aprovechamiento del espacio interior.

Interiormente, la plataforma cuenta con divisiones para poder alojar sus elementos más importantes de forma más repartida y con una adecuada distribución de pesos, ya que dentro de la misma se sitúan componentes de considerable peso para este tipo de robot, como por ejemplo la batería. La caja en Edubot está cerrada completamente excepto por su parte superior, en la que se sitúa una tapa con una bisagra en medio para poder acceder a cada una de las divisiones por separado. En cambio, en los robots Sacarino el acceso al interior de la plataforma móvil se realiza mediante unas puertas laterales con bisagras, para evitar tener que desmontar la parte superior del robot cada vez que se quiera acceder.

El material de todos los chasis es aluminio, material típico en este tipo de construcciones, que ofrece: resistencia, para soportar los pesos, vibraciones y

otros posibles problemas debidos a la movilidad; bajo precio, muy importante en general, y especialmente en proyectos de estas características; bajo peso, lo que redunda en mejores prestaciones y también menor peso en otros componentes, como los motores o la batería; y facilidad de mecanizado, lo que facilita en general adaptarlo a las formas deseadas.

En las siguientes figuras se pueden apreciar la forma y dimensiones del robot Edubot (*5.6*) y de la plataforma móvil de Sacarino (*5.7*):

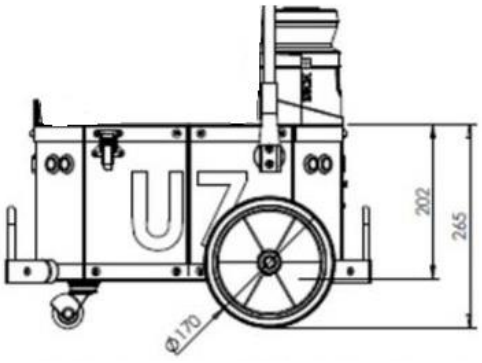

Figura 5. 6. Plataforma móvil Edubot.

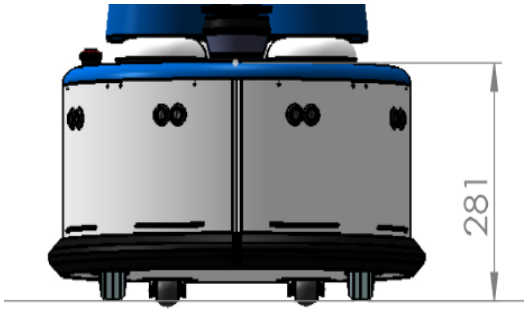

Figura 5.7. Plataforma móvil de Sacarino 2.

Uno de los puntos más importantes a considerar en el diseño de cualquier plataforma móvil es la configuración cinemática, es decir, el sistema de movimiento del robot, que se sirve de diferentes configuraciones de ruedas y grados de libertad para moverse. Existen muchas configuraciones cinemáticas distintas, para distintos tipos de robots y con diferentes niveles de complejidad y precio.

La configuración del robot Sacarino original es de tipo Sincrodrive, es decir, sus ruedas (cuatro en este caso) están controladas por dos motores, uno de tracción que las hace girar, y por lo tanto posibilita el movimiento, y otro que las hace cambiar su dirección para orientar el robot adecuadamente. Todo esto gracias al enlace entre las ruedas mediante un sistema de correas, según *figura 5.8*.

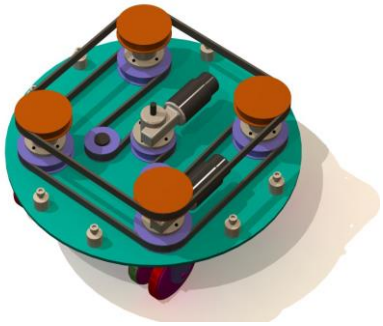

Figura 5.8. Sistema sincrodrive de Sacarino.

Sin embargo, la configuración adoptada en el robot Edubot y que sirve perfectamente a los propósitos para los que está destinado es la configuración cinemática direccional, que consta de dos ruedas motrices, situadas una a cada lado del robot y con sendos motores para su control por separado. No obstante, para mejorar la estabilidad del robot, éste tiene dos ruedas locas (sin control sobre ellas) en la parte delantera y algo más centradas.

A fecha de finalización de esta memoria aún no se tienen detalles específicos sobre la plataforma móvil de la segunda versión de Sacarino, pero por el diseño planeado, la plataforma de éste y su configuración cinemática serán muy parecidas a la del robot Edubot, como se ve en la figura 5.9.

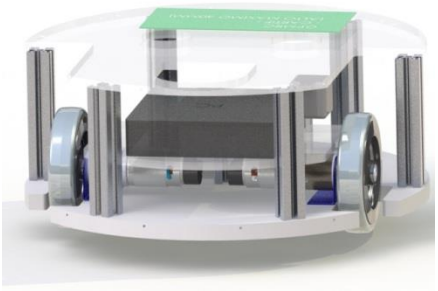

Figura 5.9. Disposición cinemática de Sacarino 2.

Los motores unidos a las ruedas motrices pueden ser de diversos tipos, también dependiendo de las especificaciones requeridas. Por un lado, los motores de corriente alterna proporcionan grandes velocidades, pero no es el caso de este robot. También están los motores paso a paso o los servomotores, pero a pesar de que ofrezcan una alta precisión, son complicados y más caros. El tipo de motor que incorpora Edubot es un motor de corriente continua, que es sencillo de utilizar, suficientemente preciso y puede otorgar velocidades considerables o velocidades más bajas pero con

un alto par, lo que le proporciona versatilidad ante posibles terrenos o situaciones difíciles.

Lo último en relación a la plataforma móvil es la alimentación de la misma y de los robots. Para alimentar el conjunto de los elementos es importante la elección de una adecuada batería, que conjugue a la vez facilidad de funcionamiento y de carga, precio asequible, autonomía razonable y que proporcione las especificaciones eléctricas deseadas para alimentar los sistemas, en este caso 12 voltios.

Debido a las diferentes exigencias de los robots expuestos, la alimentación de los mismos también será distinta. Por ejemplo, en el caso del robot original, eran necesarias 2 baterías para controlar todo, desde el movimiento de la plataforma móvil hasta el ordenador y su pantalla táctil. Sin embargo, el aumento de eficiencia de los sistemas del robot y las mejoras en la batería han posibilitado que tanto el robot Edubot como la segunda versión de Sacarino tengan sólo una única batería alimentando al conjunto.

5.2.2 Sensores

Además de los sistemas mecánicos y motores explicados, una parte muy importante de los robots son los diferentes sensores que le permiten conocer determinadas características del entorno, generalmente relacionadas con el espacio y el movimiento, que son las relacionadas con la función principal del robot como plataforma móvil.

Algunos de estos sensores son completamente necesarios y otros simplemente complementan a otros sistemas o dan unas funciones específicas pero sin las cual es posible controlar el robot igualmente.

Los principales sensores con los que cuentan los robots, en especial en relación al movimiento, que es lo importante aquí, son los siguientes:

- Codificadores incrementales: se acoplan a los motores solidarios a su eje con el objetivo de informar de su giro y, en consecuencia, de la posición concreta de la rueda en cada instante. Así, son esenciales para el correcto y preciso funcionamiento del movimiento del robot.

- Sensores ultrasónicos: se basan en la emisión y recepción de ráfagas de ondas de ultrasonidos y en la cantidad de tiempo que tardan en recibir el eco para medir distancias entre 2cm y 3m y son una forma simple pero efectiva de evitar obstáculos en su trayectoria, ya sean objetos o paredes. Su disposición y número varía según el robot: en Edubot, son ocho situados especialmente en las paredes delantera y trasera para evitar chocar con

obstáculos; en el Sacarino original, son 16 colocados por todo el perímetro lateral del robot; en la última versión de Sacarino son ocho también, de nuevo colocados en el perímetro de la circunferencia que forma la base del robot.

- Láser: aunque no sea necesario para las aplicaciones de visión artificial que se desarrollan en el proyecto, el robot tiene la posibilidad de usar un láser 2D que le sirve para situarse y moverse de manera muy precisa, generando un mapa del entorno. Es uno de los elementos que podrían ser sustituidos por un sistema de visión en tres dimensiones para mejorar el comportamiento y el precio del robot. Se encuentra en los tres robots, siempre en la misma posición: la parte delantera del robot (ver *figura 5.10).*

Figura 5.10. Plataforma móvil de Sacarino 2.

- Bumpers: son los sensores de choque que sirven en última instancia tanto para amortiguar como para avisar de un choque con un obstáculo una vez ya se ha producido. Se colocan en Edubot y Sacarino de manera semejante a como se colocaban los sensores de ultrasonidos.

El sistema de visión artificial que se va a utilizar en este proyecto estaría enmarcado en este apartado de sensores, pero al ser parte esencial del proyecto, se ha explicado por separado.

5.2.3 Estructura de control

El control de forma directa de todos y cada uno de los elementos descritos en el apartado anterior sería muy complicado, puesto que se basan en señales eléctricas muy precisas y concretas para funcionar. De ahí radica la necesidad de usar un interfaz que una esos elementos físicos con una programación más sencilla para el programador, sobre todo teniendo en cuenta que la presente plataforma robótica tiene pretensiones educativas.

Esta placa de control tiene como misión traducir las órdenes de control de alto nivel a los dispositivos de bajo nivel, y al contrario, recopilar la información de sensores y proporcionársela al control de alto nivel. Exactamente, la placa controladora utilizada es la GPMRC (General Purpose Mobile Robot Controller), modelo GPMRC6LC, diseñada específicamente para los robots utilizados por el centro tecnológico Cartif.

Figura 5.11. Controladora GPMRC.

Esta placa, vista en la *figura 5.11*, está constituida por dos microcontroladores dsPIC30F6011 en modo maestro/esclavo que gestionan la información de los elementos del robot y generan alarmas de fallos en los casos necesarios.

Sin embargo, la placa GPMRC no gestiona directamente los motores y sensores, sino que se sirve de una controladora de motores para controlar el giro de cada uno de ellos y una placa concentradora de sonars y bumpers debido a la lenta tasa de lectura de los mismos.

Por último, para la conexión de la placa con el dispositivo de control a alto nivel elegido (varios posibles según el uso que se pretenda) se utiliza un puerto serie, y además se ha incluido directamente en la GPMRC la posibilidad de controlar el robot de forma manual mediante un Joystick de tipo Playstation, que manda las órdenes sin necesidad de pasar por ningún otro elemento de control a alto nivel

De esta forma, la arquitectura física completa de Edubot (no muy distinta de la de Sacarino, excepto que este incluye más componentes) y sus distintos elementos físicos queda como se puede observar en la siguiente figura:

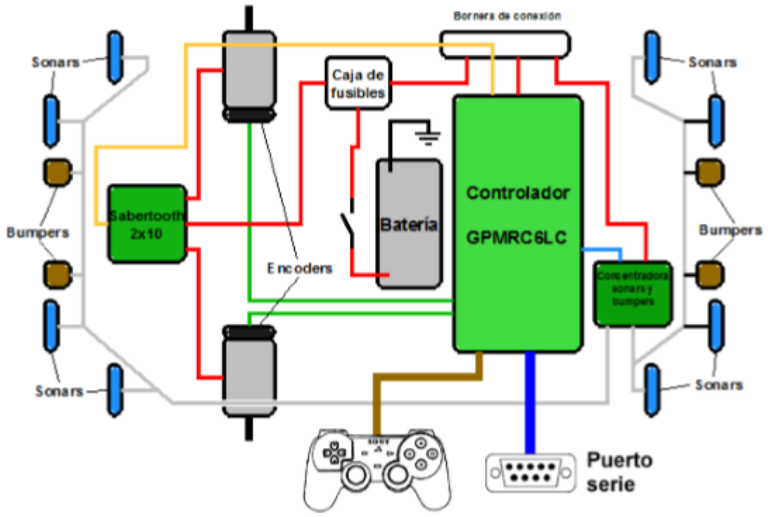

Figura 5.12. Arquitectura de Edubot.

5.2.4 Ubicación de Kinect

En el tercer capítulo se eligió un dispositivo de visión artificial, Kinect, con el que se va a trabajar en el presente proyecto, desarrollando aplicaciones de visión en tres dimensiones. A la hora de implementar el dispositivo físico en el robot, hay que tener en cuenta alguna de las características del aparato, vistas también en el apartado 3.3, en especial para que el rango de visión se adecúe a las necesidades de reconocimiento de personas y gestos. Los dos aspectos principales a tener en cuenta son la colocación física y la conexión con el resto de hardware que ya se ha hablado.

Respecto a la colocación, tanto en altura como en dirección y orientación, aunque se hablará más de ello al hacer las pruebas pertinentes en el capítulo 7, se puede dejar reflejado que para obtener la máxima naturalidad en la interacción entre robot y persona, la altura debe ser la adecuada para capturar en la imagen de la cámara a la persona y la dirección, preferiblemente de frente al propio robot, por lo que Kinect se colocaría en su parte delantera.

La conexión con el resto del hardware es sencilla, puesto que sólo requiere de una conexión USB. Aunque la controladora GPMRC disponga de conexiones para ello, la necesidad de un ordenador con suficiente capacidad de procesamiento hace que la cámara deba estar conectada directamente a él, y por tanto el resto del hardware se conecta al ordenador como se ha visto antes.

Sin embargo, otro dato importante es la alimentación de Kinect, de 12V, y que debe proveerse o bien desde un enchufe común, o bien, como ha sido el caso en alguna de las pruebas, de alguna salida de tensión disponible en el propio robot, siempre de 12V, por supuesto.

5.3 Software y programación

Tan importante como los sistemas físicos utilizados es la programación y control de un robot de forma adecuada, para hacer un buen aprovechamiento de los motores y sensores de que constan y obtener los resultados deseados. Aquí es tan relevante el uso de una adecuada plataforma de control a alto nivel como la programación de los módulos de la misma.

5.3.1 Control a alto nivel

El control a alto nivel es aquel que está más cerca de la capacidad cognitiva humana. Puesto que es la programación realizada por el ser humano directamente, debe ser más fácil de comprender y manejar adecuadamente.

Aunque los robots Sacarino, debido sobre todo a su complejidad y a la forma en la que están realizados sólo pueden incluir la primera plataforma de las que se van a listar, en realidad existen numerosas alternativas posibles para controlar a alto nivel a un robot que haga uso de la controladora GPMRC, como es el caso de Edubot, aunque esto dependerá también del uso que se le espere dar. En cualquier caso, las posibles plataformas de control a alto nivel son:

- Ordenador portátil: es la plataforma más versátil y adecuada para la mayoría de aplicaciones. Su potencia es la principal razón para su uso, pudiendo controlar (con un ordenador estándar) sin problemas ROS y los sensores y actuadores necesarios sin sacrificar velocidad. Sin embargo, su tamaño y coste pueden hacer que resulte excesivo para aplicaciones simples.

- Raspberry Pi: es un ordenador de reducidas dimensiones y de bajo coste con unas especificaciones más simples que las de un ordenador portátil pero suficientes para determinadas funciones y aplicaciones del robot.

- Arduino: es una plataforma también de bajo coste para controlar elementos hardware de forma sencilla. Puede utilizarse para el control de un

conjunto de sensores o motores, aunque cuenta con ciertas desventajas y limitaciones en la conexión con estos. Actualmente se está realizando un proyecto para controlar Edubot con Arduino, pero su capacidad de procesamiento tampoco es suficiente para funciones avanzadas.

- Joystick: por último, para situaciones que no requieran un control a alto nivel avanzado, se ha incluido la conexión directa de la placa GPMRC a un joystick de tipo Playstation (ver *figura 5.13*) de forma que se pueda controlar el movimiento del mismo directamente mediante este mando, sin pasar por un modo autónomo que requiera una programación más avanzada.

Figura 5.13. Joystick de tipo PS3.

De entre las diferentes formas de controlar Edubot que se han analizado, la elegida será el ordenador portátil, puesto que es la única que puede soportar la capacidad de procesamiento necesaria para las aplicaciones de visión artificial requeridas. Además, el propio Sacarino cuenta con un ordenador embarcado, y si se pretende utilizar la visión artificial desarrollada para él, el modo más parecido es el del portátil.

Esto queda aún más reforzado examinando algunos de los requisitos mínimos de la cámara Kinect:

- 2GB de memoria RAM.
- Procesador de 32 o 64 bits.
- Bus USB 2.0 dedicado.

Ni Arduino ni Raspberry ni otro tipo de plataformas simples podrían soportar estos requisitos, especialmente el de la memoria RAM, y es por eso por lo que se hace necesario un ordenador como plataforma de control a alto

nivel, sin posibilidad de que los programas desarrollados funcionen adecuadamente en otras plataformas con menos potencia.

5.3.2 Sistema operativo

Como tantos otros robots actuales, Sacarino y Edubot utilizan para moverse el sistema ROS explicado en el capítulo anterior, lo que posibilita que la mayoría de las aplicaciones que se desarrollen para cualquiera de los robots que utilice este sistema sean más o menos fácil de llevar a otros robots del mismo tipo, en este caso robots móviles de tipo social.

Al partir de robots ya construido y en completo funcionamiento como son estos dos, existen ya un conjunto de módulos y nodos de ROS creados para hacerlos funcionar de manera adecuada, especialmente en lo relativo al movimiento de las plataformas móviles, que es lo único que se utilizará para el trabajo actual.

Existen multitud de módulos ya programados, bien en Cartif o bien por la vasta comunidad de ROS, que se podrían utilizar junto a los que se van a describir, pero sólo unos pocos son absolutamente esenciales para el funcionamiento y el movimiento del robot. A continuación se describirán los nodos del robot Edubot, pero en Sacarino, especialmente los nodos más a bajo nivel, son prácticamente iguales.

Uno de los nodos más importantes es el nodo Eusebio, que se encarga de enviar los comandos adecuados de velocidad para mover el robot en la dirección adecuada en base a, generalmente, las órdenes del joystick. Además, se encarga también de enviar las órdenes de parada adecuadas si algún objeto o pared está demasiado cerca acorde a la información de los sonars o bumpers.

El nodo Gpmrc_node se encarga de leer la información enviada por Eusebio para realizar el movimiento y transformarla en valores que pueda leer el hardware a bajo nivel correctamente.

Por último, el nodo Serial_port se encarga de posibilitar la comunicación entre el sistema de control a alto nivel, en este caso el portátil, y el propio hardware (motores, sensores…).

En la *figura 5.14* se observa un esquema básico con la interconexión de estos nodos y sus comunicaciones.

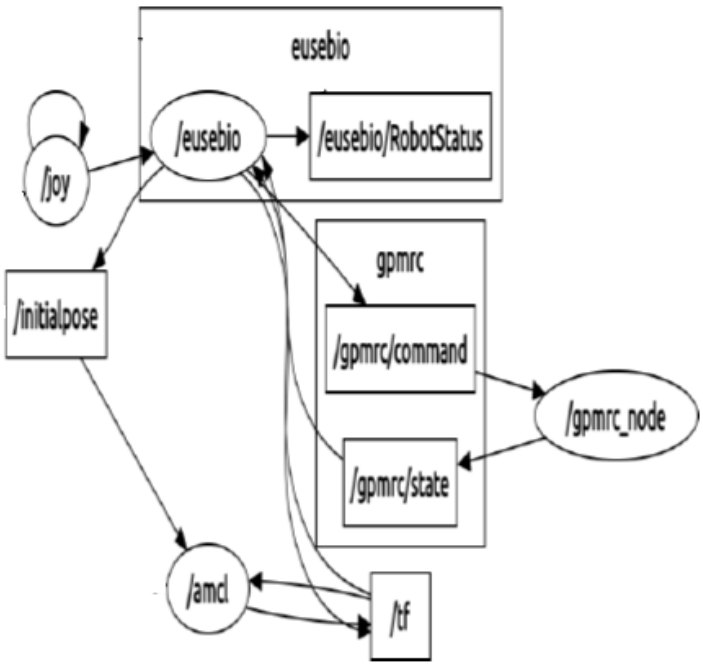

Figura 5.14. Nodos preexistentes en el robot.

Partiendo de esto, las aplicaciones que se desarrollen deberán adaptarse al esquema anterior intentando modificar sus nodos lo menos posible, o directamente sin añadir modificación alguna, trabajando con los datos tal cual existen en el robot original. No debería ser esto ningún problema puesto que el sistema de visión artificial es completamente externo y ajeno a las configuraciones de los nodos anteriores.

Sin embargo, para aquellas aplicaciones que en base a la información del sistema de visión modifiquen el comportamiento móvil del robot será necesario que las órdenes de desplazamiento se comuniquen en un formato adecuado que no altere la forma de trabajar de los nodos anteriores.

Por ello, dichas órdenes se darán en formato de tipo joy, es decir, simulando mensajes del joystick, de forma que el nodo Eusebio lea estos mandatos como si fueran órdenes normales de movimiento con el joystick, y así se pueda evitar modificarlo, lo cual conllevaría aún más problemas y complicaciones.

Capítulo 6:

Diseño y desarrollo de los sistemas de reconocimiento

En este capítulo se van a detallar los procesos para la realización de las distintas funciones objetivo del robot, así como la unión entre ellas para la consecución de las metas fijadas. Para comprender mejor todo lo que aquí se va a exponer, conviene estar familiarizado con el sistema operativo ROS y algunos de sus conceptos más básicos, explicados ya en el capítulo correspondiente.

ROS, mediante su diseño modular, posibilita la implantación de manera muy simplificada de las funciones deseadas, pudiendo activar o desactivar unas u otras según la situación o el usuario lo requieran.

6.1 Configuración inicial.

Como punto de partida, debe ser tenido en cuenta que las funciones que se van a implementar son un añadido a una programación ya existente previamente. Esto significa que ya hay un conjunto de nodos funcionando en todo momento y que se encargan del movimiento del robot, como se vio en el capítulo anterior y se reproduce de nuevo aquí en la *figura 6.1.*

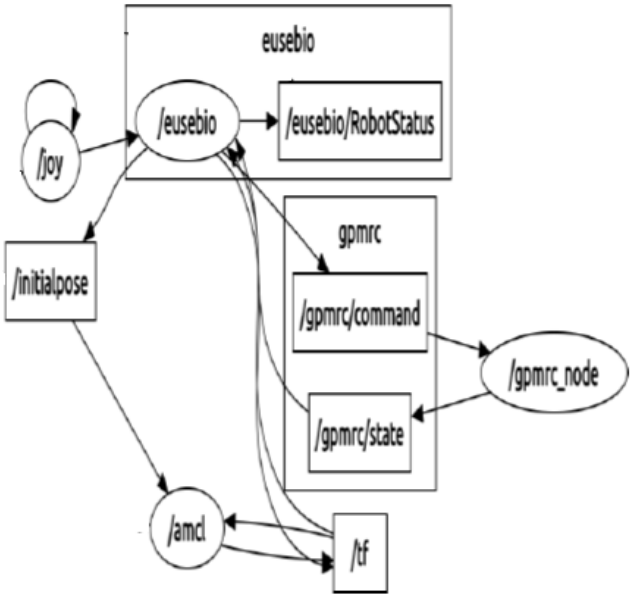

Figura 6.1. Nodos del robot preexistentes.

Lo que se va a hacer es añadir un conjunto de nodos que realicen las funciones de visión artificial sobre estos nodos ya completos relativos al movimiento (en este caso de Edubot, pero que para el caso de Sacarino o de otro robot del tipo plataforma móvil son equivalentes).

Como el sistema de visión artificial que se está utilizando es Kinect, ya existe un paquete de nodos y aplicaciones diseñados para que interactúe con ROS, y de hecho se instala junto al sistema operativo (en la versión utilizada, Groovy) de forma automática en forma de 3 paquetes: *openni_camera, openni_launch* y *openni_tracker.* En el caso de haberlos desinstalado o de preferir instalarlos manualmente, el comando para hacerlo sería:

```
Sudo apt-get install ros-hydro-openni-*
```

Sin embargo esto no es suficiente para el correcto funcionamiento de todas las aplicaciones relacionadas con estos paquetes, y es probable que se produzcan numerosos errores. Es necesario instalar también los drivers o controladores *openni* para hacer funcionar Kinect adecuadamente, porque sin ellos es detectada como una cámara estereoscópica más. Aunque puede ser utilizada como tal, no se tiene acceso a muchas otras funciones y programas desarrollados específicamente para Kinect, que como ya se ha dicho, es una de las principales razones para utilizar este dispositivo. Estos drivers se adjuntan con los programas realizados, y tan sólo necesitan ser ejecutados e instalados como otros drivers cualesquiera para Ubuntu.

Una vez realizadas estas configuraciones iniciales, ya deberían estar preparados para usarse todos los nodos y aplicaciones necesarios inicialmente para el uso de la visión artificial.

En concreto, el paquete de más utilidad genérica para tratar los distintos datos es *openni_launch*, que contiene varios archivos ejecutables que lanzan conjuntos de nodos para tratar distintos datos. Por ejemplo, contiene el ejecutable *depth.launch* que lanza los nodos y drivers encargados de realizar el mapa de profundidad, o el ejecutable más importante, *openni.launch*, que lanza todos los nodos y topics referentes a la cámara, incluyendo la profundidad, pero también las imágenes de la cámara y otros útiles para su utilización.

Sin embargo, en el transcurso de este proyecto la utilidad de este paquete se va a centrar en la representación visual de los distintos tipos de imágenes de la cámara, sin ser estrictamente necesario para las funciones desarrolladas aquí.

El otro paquete, que en este caso sí va a ser utilizado, es *openni_tracker*, que va a ser descrito a continuación como el primero de los sistemas desarrollados.

6.2 Sistema de reconocimiento de personas.

La primera de las funciones que incorpora el sistema de visión artificial es el reconocimiento de personas. El sistema reconoce hasta un máximo de 15 usuarios a la vez y es capaz de monitorizar en tiempo real el movimiento y cada una de las partes del cuerpo de todos ellos, aunque la fiabilidad del sistema es superior cuando sólo hay uno o dos delante de la cámara, no más, ya que por limitaciones de procesamiento es posible que la velocidad del sistema sea mucho menor y se produzcan distorsiones y ralentizaciones en el esqueleto captado.

6.2.1 Inicio y calibración.

El nodo encargado de esta monitorización es *openni_tracker,* que se puede encontrar dentro del paquete *openni_tracker*. El comando para lanzarlo es:

```
rosrun openni_tracker openni_tracker
```

Una vez hecho esto, el sistema queda a la espera de que aparezcan usuarios delante de la cámara. En el momento en el que detecta un nuevo 'usuario' (puede no ser una persona incluso, simplemente algo en movimiento) dentro del ángulo de visión, empieza un proceso de calibración en el que, mediante el conjunto de funciones que incorpora el módulo utilizado, se detecta a un usuario y su esqueleto con un conjunto de articulaciones y elementos.

Esta calibración es de hecho superior a las calibraciones de otras aplicaciones o versiones anteriores de esta misma, porque no requiere ninguna pose específica para rastrear al usuario, tan sólo con su propio movimiento y actuando de manera natural delante de la cámara se puede conseguir una correcta calibración. Podría ser aún mejor, ya que por ejemplo, no es capaz de capturar bien a una persona cuando esta se encuentra inmóvil delante de la cámara, debe hacer algún movimiento ligero al menos, lo que también es una limitación del sistema, aunque aun así sea menos rígido que en las versiones previas.

Más concretamente, anteriores versiones de este paquete tracker requerían que el usuario adoptara una pose especifica denominada *pose Psi* (ver en la *figura 6.2*). Si bien es una pose fácil de adoptar, hacía menos natural la interacción inicial con el robot en el caso de que quisiéramos ser detectados.

Figura 6.2. Pose Psy.

Otro de los puntos fuertes del algoritmo de rastreo es lo innecesario de que la cámara capte el cuerpo humano en su totalidad para calibrar su posición y rastrearlo en tiempo real. Puede parecer poco importante pero, como ya se ha visto, la colocación de Kinect en el robot y la distancia de interacción humano-robot es limitada, y cuanto mejor sea la captación de los movimientos sin tener que estar completamente dentro del rango de visión de Kinect, mucho mejor.

El algoritmo de rastreo, además de suponer la posición de las articulaciones y miembros del cuerpo que no se encuentran dentro del campo visual (de acuerdo a la lógica de la forma humana), es capaz de detectarlas a posteriori cuando entren en dicho campo aun si no se han utilizado en la calibración inicial.

Esta calibración, mientras no se haya realizado completa y correctamente, es recurrente, es decir, muestra un mensaje de error y vuelve a comenzar. Pero una vez que se ha conseguido calibrar y seguir los movimientos del usuario de forma fidedigna, este queda monitorizado mientras no salga del ángulo de visión durante algunos segundos seguidos. No obstante, el sistema sigue atento por si otro posible nuevo usuario apareciera delante de la cámara, para aplicar al mismo de nuevo la calibración. Incluso las detecciones y calibraciones pueden realizarse de manera simultánea para varios usuarios sin resentirse el tiempo de procesamiento ni la fiabilidad de las mismas, como se aprecia en la *figura 6.3*.

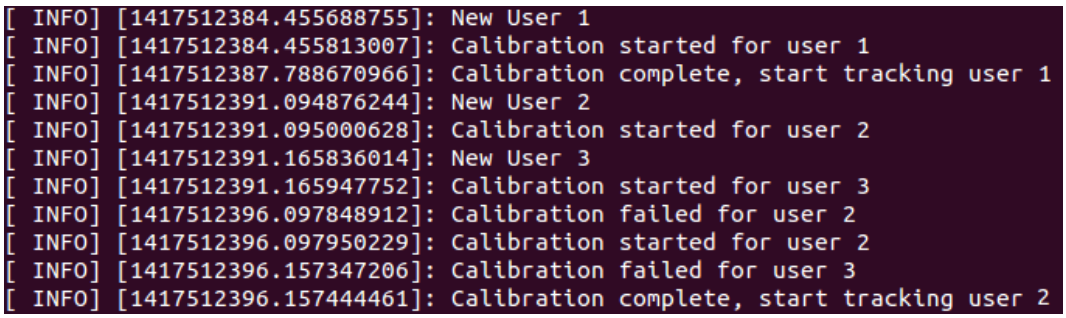

Figura 6.3. Calibración de usuarios.

6.2.2 Modelo del usuario

Lo obtenido una vez que se realiza la detección y calibración de manera satisfactoria es el modelo en tres dimensiones de varios puntos del usuario. Más específicamente, se sigue el movimiento en tiempo real de algunas de las articulaciones y miembros más importantes de la persona: cabeza, torso, codos, manos, rodillas, pies… que se publican como tf (*transformed frames*). El listado completo de estos tf es el siguiente (el número 1 junto a cada tf se refiere al usuario, puede ir de 1 a 15 dependiendo del usuario del que se quiera obtener):

/head_1
/left_elbow_1
/left_foot_1

```
/left_hand_1
/left_hip_1
/left_knee_1
/left_shoulder_1
/neck_1
/right_elbow_1
/right_foot_1
/right_hand_1
/right_hip_1
/right_knee_1
/right_shoulder_1
/torso
```

Todo esto forma parte de un árbol en el que todos los tf, independientemente del usuario del que forman parte, son hijos de un sistema de referencia padre con su origen en el centro de la propia cámara Kinect, denominado */openni_depth_camera*.

La *figura 6.4* muestra esto, aunque por el gran número de tf hijos la imagen no se muestran todos, pues sería imposible de representar, además de completamente innecesario.

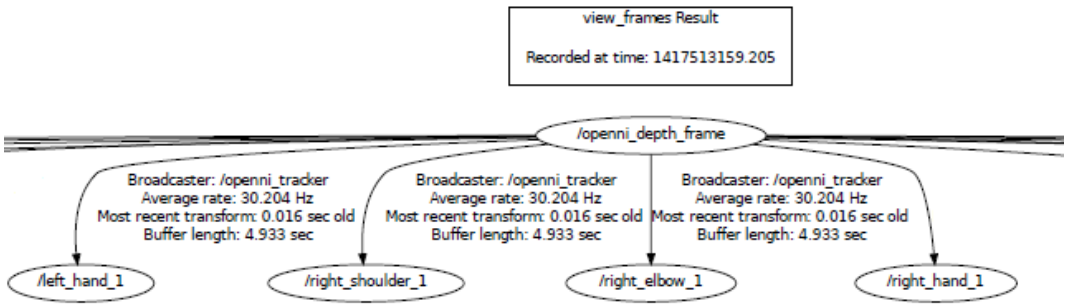

Figura 6.4. Árbol de transformed frames.

Cada una de estas transformaciones contiene varios datos útiles para trabajar con las mismas. Como se explicó en el capítulo sobre ROS, los *transformed frames* funcionan de manera similar a cualquier otro dato intercambiado en ROS, y se publican en el topic /tf como un mensaje normal con diferentes campos (se muestra en la *figura 6.5* el mensaje mandado correspondiente a una de las articulaciones, pero en cada iteración del nodo se envía una de estas por cada articulación o elemento captado, haciendo un total de 15).

```
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 1417513657
        nsecs: 243417554
      frame_id: /openni_depth_frame
    child_frame_id: /right_foot_1
    transform:
      translation:
        x: 0.994636373093
        y: -0.254436455751
        z: -0.863648306826
      rotation:
        x: 0.499999999997
        y: 0.500001836603
        z: 0.499999999997
        w: 0.499998163397
```

Figura 6.5. Mensaje de tipo tf.

Una vez obtenido esto, se puede comenzar a trabajar con estos tf y sus valores en cualquier otra aplicación que lo requiera. Para ayudar a su visualización, además, se puede utilizar el software rviz incorporado en ROS y que muestra la distribución espacial del árbol de transformaciones, con las uniones entre el padre (*/openni_depth_frame*) y los hijos (ver *figura 6.6*).

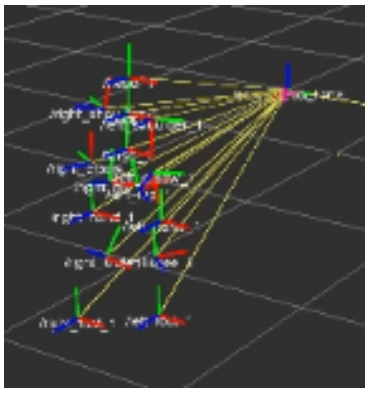

Figura 6.6. Distribución espacial de los tf.

6.3 Sistema de reconocimiento gestual.

Otra de las aplicaciones en las que se ha trabajado es el reconocimiento de gestos o posiciones del cuerpo, que también podía encontrarse en paquetes o módulos ya existentes para ROS y que podían ser adaptados, pero se ha preferido desarrollar un módulo propio desde cero.

Las razones para esto estriban en la alta complejidad de los paquetes ya existentes, innecesaria a todas luces y que además podrían causar ralentizaciones y problemas de compatibilidad, sobre todo teniendo en cuenta que estaban desarrolladas para versiones anteriores del sistema operativo utilizado en el robot, lo que también era un obstáculo a su uso. Uno de los más famosos programas de reconocimiento de gestos está desarrollado por el MIT [15], pero para una versión muy anterior de ROS, por lo que no se ha utilizado.

6.3.1 Inicio y adquisición de datos.

El nodo desarrollado para esto se encuentra dentro del módulo gestos, llamándose también así. Para mejores explicaciones, posteriormente se hablará sobre el lanzamiento de todos los nodos y paquetes, pero el comando para lanzarlo por separado sería:

```
rosrun gestos gestos
```

En cualquier caso, al ejecutar el nodo comienza un proceso de lectura de las articulaciones del cuerpo obtenidas con *openni_tracker.* Por eso es indispensable ejecutar dicho nodo previamente y que haya obtenido la posición del usuario inicial para el funcionamiento de este. No obstante, debido a que el nodo es recursivo y se ejecuta constantemente en bucle, se puede ejecutar el tracker a posteriori, pero el nodo gestos no tendrá ninguna utilidad hasta que se haya detectado al menos una persona con *openni_tracker.*

Una vez que se sigue el movimiento de una persona con el nodo correspondiente, el código de gestos se encarga de leer por separado todas las articulaciones del usuario correspondiente y obtener de las mismas la posición exacta respecto a la propia cámara Kinect en las tres dimensiones, xyz.

Para esto cobran importancia las funciones que se encargan de leer y almacenar en un objeto (recordando que en todo el programa se está trabajando en C++) los valores de estas distancias.

Aquí cabe destacar de nuevo a qué distancias corresponden cada uno de los ejes o coordenadas (*figura 6.7*):

- Coordenada x: distancia Kinect-usuario.
- Coordenada y: distancia horizontal perpendicular a x (ancho usuario).
- Coordenada z: altura.

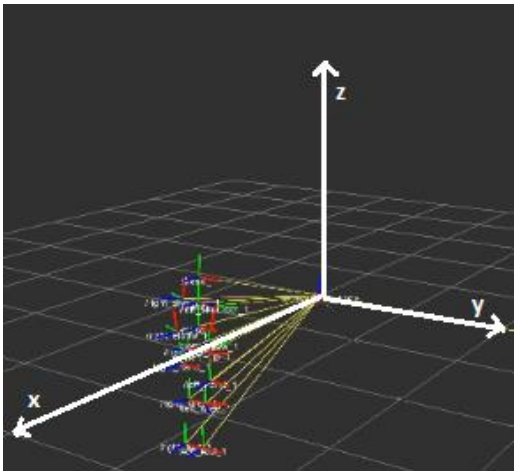

Figura 6.7. Coordenadas de los tf.

Este es otro punto en el que cobra importancia la colocación de Kinect, y especialmente su inclinación, puesto que si está inclinada, el usuario se encontrará inclinado en su modelado y articulaciones, y las distancias no se corresponderán exactamente con las coordenadas según lo esperado, sino con cierto ángulo de inclinación. Se entrará en más detalles sobre esto en los resultados, en el capítulo siguiente.

6.3.2 Interpretación de gestos.

Después de tener almacenadas las variables de cada una de las articulaciones y elementos del cuerpo, se pueden tratar de diferentes maneras para ser capaz de leer los gestos o posiciones de una persona, incluso determinados movimientos simples.

Para llevar a cabo este proceso (partir de las posiciones de los elementos del cuerpo y obtener gestos), se pueden utilizar diversos métodos más o menos avanzados según el tipo de aplicación que se espere desarrollar:

- Evaluación de diferentes posiciones corporales con las coordenadas de los elementos: de forma simplificada, se encarga de comparar la posición de una articulación respecto a otra para determinar una posición corporal simplificada. Por ejemplo, si hombro-codo-mano tienen la misma coordenada z, posición "brazos en cruz".

- Evaluación en exclusiva de la posición de las manos: semejante al anterior método pero utilizando exclusivamente la posición de las manos debido a que son los elementos corporales que más comúnmente estamos acostumbrados a utilizar para la interacción y están sujetas a una fácil interpretación. Por ejemplo, una mano sensiblemente por delante (es decir, con coordenada x inferior) del resto del cuerpo, puede significar "posición stop".

- Variaciones en el tiempo en las posiciones anteriores: es decir, tener en cuenta la posición que ha tenido un determinado miembro anteriormente para observar la variación que ha experimentado. Un ejemplo de esto serían los movimientos "mano subiendo" o "mano bajando".

- Recta de regresión: para tener en cuenta, por ejemplo, la rectitud de los brazos, se puede observar el coeficiente de correlación entre las articulaciones de hombro, codo y brazo con sus coordenadas x e y.

- Métodos avanzados de comparación de posiciones: existen métodos matemáticos más sofisticados para comparar la posición en un determinado momento de un conjunto de elementos con sus coordenadas espaciales y una posición previamente almacenada en memoria. Unos ejemplos de herramientas matemáticas que podrían utilizarse para esto son la distancia de Mahalanobis [30] y la distancia de Hausdorff [10].

En la implementación final elegida se han adaptado los tres primeros métodos por ser los más simples y que mejores resultados han dado. No confundir el segundo método con la evaluación de las posiciones de los elementos de una mano (los dedos) porque la precisión de Kinect no es suficiente para ello. De haber querido realizarse, se debería haber utilizado otro tipo de sensor más preciso en este tipo de aplicaciones como Leap Motion.

El método de la recta de regresión ofrece buenos resultados, pero obligaría al usuario a mantener una posición muy estricta y normalmente antinatural,

por lo que se ha desechado. Y los métodos más avanzados resultan demasiado complicados para las posiciones que aquí se pretenden utilizar, y a menudo serán demasiado elaborados para la precisión que Kinect puede otorgar, por lo que también han sido rechazados.

De esta forma, y teniendo en cuenta los métodos elegidos anteriormente y el campo de visión del dispositivo (que ya se ha evaluado en el capítulo anterior), los gestos, posiciones y movimientos que se han elegido determinar, con imágenes representativas de ellos en las figuras 6.8 a 6.1x, son:

- **Brazo izquierdo en cruz** (*figura 6.8*): con el brazo izquierdo extendido y alineando hombro, codo y mano.

Figura 6.8. Brazo izquierdo en cruz.

- **Brazo derecho en cruz** (*figura 6.9*): con el brazo derecho extendido y alineando hombro, codo y mano.

Figura 6.9. Brazo derecho en cruz.

- **Brazos en cruz** (*figura 6.10*): para el caso de combinación de los dos gestos anteriores

Figura 6.10. Brazos en cruz.

- **Stop** (*figura 6.11*): gesto muy común para ordenar la parada, con el brazo extendido hacia delante.

Figura 6.11. Stop.

- **Brazos en alto** (*figura 6.12*): los dos brazos levantados sobre la cabeza.

Figura 6.12. Brazos en alto.

- **Mano acercándose o alejándose:** aunque no sea visible en una imagen, cuando una mano se acerca o se aleja del resto del cuerpo del usuario, también es detectado.

- **Mano subiendo o bajando:** igual que el anterior, no es visible en una imagen estática, pero cuando una mano sube o baja independientemente de su posición en los otros dos ejes, también puede ser detectado.

Para determinar esto, no sólo se ha tenido en cuenta que se produjera la posición determinada, sino la duración en el tiempo de esta. Debido a que el algoritmo se ejecuta varias veces en un mismo segundo, sería fácil que una posición de transición o simplemente una posición natural en un instante determinado se confundiera con una de las elegidas como interacción, por lo que se necesita que la misma posición se repita durante varios ciclos del programa para evitar esto. La repetición necesaria para reconocer la posición correcta se puede establecer y variar, según la frecuencia del programa, tiempo de procesamiento o por observación de problemas en el uso del reconocimiento de gestos en diferentes plataformas.

6.3.3 Aplicación de los gestos.

Como resultado final, una vez que se ha determinado que el usuario se encuentra realizando una determinada posición o movimiento, se puede utilizar el mismo en el propio código del programa o enviarlo mediante un topic correspondiente para que puedan hacer uso de la información otros nodos, posiblemente modificando su comportamiento en base a los gestos, que es al final para lo que se realiza esto.

Una versión simplificada del diagrama de flujo que sigue el proceso, donde se puede apreciar mejor el funcionamiento del algoritmo, se encuentra en la siguiente *figura 6.13*:

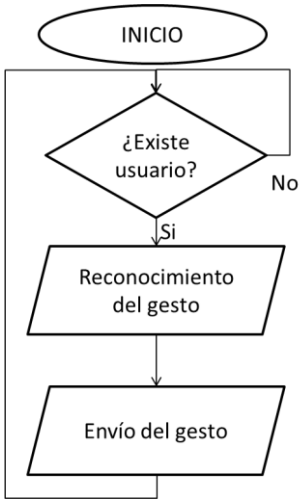

Figura 6.13. Diagrama de flujo simplificado.

6.4 Sistema de seguimiento.

Una aplicación para la que no se ha utilizado tanto Kinect, sino el propio robot y su funcionamiento como plataforma móvil, es el sistema de seguimiento y movimiento basado en gestos, que adecúa la velocidad y movimiento de los motores a la posición en el espacio y los gestos o movimientos que realiza el usuario.

6.4.1 Inicio y lectura de la posición del usuario.

De nuevo en el caso del presente nodo, se utilizan los resultados del nodo *openni_tracker* para detectar al usuario, por lo que debe ejecutarse ese antes. E igual que en el caso anterior, si no está ejecutado no sucederá nada, pero empezará a funcionar en el momento en el que se ejecute en segundo plano dicho nodo.

Además, debido a que se utiliza el movimiento del robot en el seguimiento, es indispensable tener encendido el mismo, y funcionando los nodos relativos al movimiento del mismo: *eusebio*, *gpmrc_node* y *serial_port*, así como *joy_node*, ya existente en ROS, y que se utiliza también para controlar el movimiento del robot.

El nodo desarrollado para esto se encuentra dentro del módulo o paquete gestos, llamándose seguimiento, por lo que para el ejecutarlo se debe utilizar el comando:

```
rosrun gestos seguimiento
```

Aunque en la práctica este nodo no se puede ejecutar por separado, igual que el anterior, y lo más común es lanzarlo en un ejecutable.

Una vez hecho esto comenzará la ejecución del programa de seguimiento, que dependiendo de si recibe gestos del usuario o no, actuará de una manera u otra.

La forma más simple de funcionamiento del programa es realizando exclusivamente un seguimiento del usuario. Para ello, al igual que se realiza y se ha explicado en lo relativo al programa de reconocimiento gestual, el programa lee e interpreta los datos de *openni_tracker* relativos a las articulaciones para leer la posición concreta del usuario.

Todo se realiza de la misma manera que en el programa anterior, pero simplificadamente. Puesto que el objetivo es que el robot siga al usuario, no hace falta seguir y tener en cuenta cada una de las articulaciones, con una de ellas debería ser suficiente.

De entre todas las articulaciones que se podrían elegir para realizar el seguimiento, hay obviamente algunas más útiles o normales que otras. No tendría sentido, por ejemplo, seguir el movimiento de un codo o una mano si se pretende seguir al usuario y tenerle lo más centrado en pantalla posible. Tampoco tendría sentido seguir un elemento muy bajo del cuerpo humano, puesto que como ya se vio en el apartado de colocación de Kinect, hay altas probabilidades de que se salga del campo de visión, por lo que el seguimiento se haría impreciso.

Por eso, lo más eficaz es tomar como referencia de seguimiento un elemento central del cuerpo humano. Los miembros más centrales de los que se proporciona la posición son cabeza, cuello y torso. En principio, entre estos tres no debería haber ninguna diferencia, puesto que están alineados y la altura z no es tenida en cuenta para el seguimiento, sólo la posición en el plano xy, el del suelo. Sin embargo, se ha decidido utilizar el torso para realizar el seguimiento, al ser el miembro que más centrado se encuentra de todo el cuerpo, por lo que es el que tiene menor probabilidad de quedar fuera del campo de visión de Kinect.

6.4.2 Seguimiento.

Una vez leída la posición, lo que se obtiene son las coordenadas x e y del usuario respecto a un sistema de referencia con origen en la propia Kinect, es decir, la distancia en ambos ejes del robot al usuario.

Sin embargo, el robot al moverse trabaja de otra manera. Su movimiento viene dado por dos velocidades distintas, una angular y otra lineal, como ya se ha visto anteriormente, y es el nodo Eusebio el encargado de transformar estas dos velocidades en dos velocidades distintas nuevas, una para cada motor con que cuenta el robot.

Esto significa, que las coordenadas en ejes cartesianos x e y de la articulación conviene transformarlas en coordenadas polares, para obtener una distancia y un ángulo de giro. Esto se realiza mediante las ecuaciones siguientes:

$$d = \sqrt{x^2 + y^2} \qquad \theta = \tan^{-1}\left(\frac{y}{x}\right)$$

siendo *x* e *y* las coordenadas originales, d la distancia entre el origen y el punto objetivo y θ el ángulo de giro.

Una vez implementadas las anteriores ecuaciones, se consiguen dos cosas: primero, que la velocidad lineal del robot sea superior cuanto más alejada se encuentre la persona, e inferior a medida que la distancia robot-usuario disminuye; segundo, que cuanto más desplazada esté una persona del centro del campo de visión de la cámara, mayor sea la velocidad angular, con el propósito de corregir desviación.

Es indispensable realizar un inciso aquí. Como se verá en las pruebas realizadas y sus resultados (capítulo siete), no ha habido ningún problema a la hora de controlar el robot, puesto que éste tiene una velocidad relativamente baja, tanto lineal como angular. Sin embargo, en el caso de querer aumentar esta velocidad puede hacerse necesario un control por realimentación (PID o similares) debido a oscilaciones y otros problemas típicos de este tipo de movimientos.

Volviendo a las velocidades resultantes anteriores, estas no son suficientes por sí mismas. El producto de esto no está entre los valores más adecuados para la velocidad del robot, y habrá que aplicar un factor de corrección para dar una velocidad adecuada al mismo. Además, este factor de corrección se puede utilizar para modificar la velocidad en general del robot de manera fácil sin cambiar numerosos parámetros en todo el programa.

Como ya se explicó en el apartado correspondiente al robot, éste no funciona dándole las velocidades directamente, sino que el nodo Eusebio, o el equivalente en el caso de utilizar otro robot, como Sacarino, debe recibir los valores de las mismas en un mensaje de tipo joystick como el de la *figura 6.14* de la página siguiente, por lo que hay que aplicar una última transformación en los valores de las velocidades para que se encuentren entre estos valores.

Debido a que es el aspecto más susceptible de cambio dependiendo del robot utilizado y de la forma que tenga de moverse, y que no siempre funciona como aquí se dice, se especificará un poco más en el capítulo de resultados viendo las diferencias entre uno y otro y el robot correspondiente.

```
header:
  seq: 155
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
axes: [-1.0, 0.2018004208803177, 0.0, 0.0, 0.0, 0.0]
buttons: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
```

Figura 6.14. Mensaje de tipo joy.

Este último tramo de la implementación es el más susceptible de cambio dependiendo del robot utilizado y su forma de control. Por ejemplo, hay robots que no se controlan con mensajes de tipo *joy*, sino con otro tipo de mensajes o dando valores directamente a sus motores. Sin embargo, en las pruebas realizadas se ha utilizado este método en gran medida.

Para terminar con el seguimiento, una última observación. El robot final estará programado para que no pueda comenzar el seguimiento hasta que no se le indique el gesto oportuno, pero se podría hacer que comenzara automáticamente aún sin haber detectado al usuario. También puede suceder que la Kinect dejara de detectar correctamente al usuario en medio del seguimiento.

Para evitar un comportamiento errático del robot en estos dos casos se ha añadido un pequeño mecanismo de seguridad por el que si los valores de referencia de la posición del usuario son nulos (aún no se ha detectado) o son exactamente igual a los de la iteración anterior (se ha perdido la posición después de haberlo detectado y llevar un cierto tiempo siguiéndole), el robot se para y espera a la llegada del correspondiente usuario. De no incluir esto, el robot se limitaría a dar vueltas sobre sí mismo, que puede ser un comportamiento satisfactorio también, porque buscaría al usuario perdido,

pero en la práctica se ha visto que era ineficiente y un gasto innecesario de energía.

6.4.3 Gestos y acciones en el robot.

Se ha descrito el seguimiento del robot a la persona, pero se pueden añadir modificaciones a esto último para mover el sistema de distintas maneras o darle órdenes mediante gestos.

Como se vio en un apartado previo, el reconocimiento de gestos se encarga de enviar, mediante el modelo propio de ROS de editor/suscriptor, mensajes con los gestos realizados. Gracias a ello, el nodo de seguimiento es capaz de recibir estas órdenes y modificar su comportamiento de acuerdo a ellas. Por supuesto, para esto es de nuevo estrictamente necesario que el nodo de reconocimiento de gestos se esté ejecutando a la vez.

Esto se realiza gracias a la lectura constante, en cada una de las iteraciones del programa (recordando que cada uno de los nodos se ejecuta cíclicamente), del topic por el cual se reciben los mensajes con los gestos detectados, llamado */gestos*. Una vez leídos y reconocidos por el programa, los gestos activan determinados movimientos o acciones en el robot, o se modifican algunos aspectos del seguimiento ya existente y que se esté realizando a la vez.

El listado de gestos que modifican el comportamiento del robot en la implementación final realizada, así como sus resultados en él, son los siguientes:

- **Brazos en cruz o mano acercándose:** el robot comienza el seguimiento del usuario (en concreto, de su torso) a una distancia estipulada. En este caso la distancia marcada es de 0,7 m.

- **Brazo izquierdo o derecho en cruz:** el robot comienza el seguimiento de la mano izquierda o derecha del usuario. Esto está realizado para mover el robot en alguna otra dirección, no con intención de seguimiento alguno.

- **Stop o mano alejándose:** parada inmediata del movimiento del robot en el punto en el que se encuentre.

- **Mano subiendo o bajando:** modificación de la velocidad, aumentándola o disminuyéndola respectivamente.

No obstante, los programas desarrollados han sido estructurados de manera que resulte sencillo modificar esto para cambiar los gestos que modifican el comportamiento y las acciones que realizan, por lo que queda abierta la posibilidad de cambiar los ya existentes o añadir más movimientos. De hecho alguno de los gestos no se ha utilizado, quedando a disposición del programador del mismo para realizar acciones concretas en el robot con ellos.

Para terminar, hay que destacar el comportamiento del robot en las transiciones entre diferentes gestos o acciones. La implementación, susceptible de cambios en el caso de preferir otra cosa, está realizada de tal forma que se conserva la acción anterior hasta que se realiza el gesto correspondiente a una nueva acción.

Pero además, la sensibilidad ante los distintos gestos es distinta. Se ha dado prioridad y una sensibilidad mayor al gesto de parada, como es obvio, para intentar evitar choques y problemas parecidos (sin ser tan urgente como para pulsar el botón de emergencia). Otros, sin embargo, tienen una sensibilidad menor y pueden ser algo más complicados de detectar, como el aumento de velocidad. También hay condiciones adicionales para algunos tipos de acciones, como por ejemplo, de nuevo, aumentar la velocidad, que debe tener el robot en movimiento previamente para funcionar.

6.5 Integración y descripción general de la aplicación

Si bien, como ya se ha mencionado, los tres sistemas anteriores pueden funcionar independientemente uno de otro, han sido diseñados de tal forma que interactúen entre sí sin problemas. El uso de topics o tf en ROS facilita mucho la realización de funciones por separado y poder utilizar cada una de ellas para diferentes cometidos.

El primer sistema de todos es el detector de personas, *openni_tracker*, cuyos resultados son los *transformed frames* o tf de los distintos usuarios que se pongan delante de la cámara. Así, la entrada de este nodo sería la imagen recibida por la cámara y las salidas serían las distintas posiciones de los miembros del usuario dados por el topic especial */tf*. No es como tal una comunicación editor/suscriptor como la mayoría de comunicaciones en ROS, pero se pueden utilizar los resultados publicados por este nodo para otras funciones o actividades del robot. Un ejemplo de ello para un robot humanoide completamente funcional (es decir, con brazos y piernas

completamente móviles) podría ser imitar la posición y gestos adaptando las articulaciones y eslabones del robot a los de una persona.

En segundo lugar, el sistema de gestos es el que se puede considerar como más versátil para ser adaptado para otras tareas del robot. Usa como entradas los tf de las articulaciones recibidos de *openni_tracker* a través del topic /tf y como salida los gestos a través de su topic correspondiente, /gestos. Tanto los gestos estáticos como los que detectan movimiento en tiempo real son susceptibles de ser usados de otras formas, como por ejemplo, para controlar la pantalla sin tocarla o para ordenar movimientos al robot sin necesariamente tener que seguir y mantener a la vista a la persona.

Por último, el tercer sistema es el de seguimiento, que se encarga de dar movimiento al robot según unas premisas establecidas, que también puede variar para ajustarse a nuevos usos o actuar de manera independiente a los gestos. Las entradas de este nodo, de la forma en que está desarrollado son, por un lado, las posiciones de algunas partes del usuario, dadas por el topic /tf

El conjunto de estos tres sistemas funcionando es el que finalmente se implementa en el robot, quedando un grafo de nodos como el de la siguiente figura:

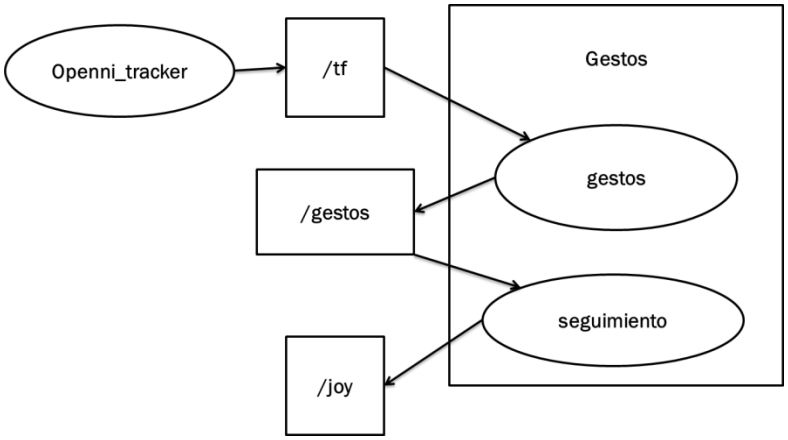

Figura 6.15. Grafo de nodos y topics de los sistemas.

Estos sistemas se unen a los que existían ya en completo funcionamiento en el robot y se expusieron en el inicio de este capítulo.

Las posibilidades de combinación del funcionamiento de los distintos nodos es muy variada, siendo éstas:

- Funcionamiento de *openni_tracker* únicamente: obviamente por sí mismo esto no realiza nada, pero se puede utilizar la base e información que

otorga openni_tracker para un uso distinto, añadiendo una nueva programación en otro nodo o simplemente para tareas de depuración.

- Funcionamiento de *openni_tracker* y *gestos*: como ya se ha dicho, las utilidades de esto serían detectar los gestos pero sin tener repercusiones en las acciones o movimiento del robot.

- Funcionamiento de *openni_tracker* y *seguimiento*: aquí de nuevo se puede tener la opción de utilizar la detección de personas para el seguimiento del robot de forma ininterrumpida o sin modificar sus acciones en base a los gestos, sino utilizando otros sistemas, como la voz o la pantalla táctil.

Es obvio que el sistema de reconocimiento de gestos sin el de personas ejecutado previamente no tiene ninguna utilidad, e igual sucede con el sistema de seguimiento, por lo que estos dos sistemas por separado no tienen ninguna utilidad. Podría decirse que la base de todo es openni_tracker y a partir de ahí los nodos se pueden añadir para realizar diferentes funciones.

No obstante, como en la implementación del robot se ha hecho el desarrollo explicado previamente, con todos ellos funcionando a la vez, es en la que tiene más relevancia explicar el modo de ejecución.

6.6 Configuración y ejecución de la aplicación.

En el capítulo tercero se explicaron brevemente los mecanismos y formas que tenía ROS de desarrollar y ejecutar los nodos, y en el apartado anterior se ha visto que los nodos gestos y seguimiento se encuentran en el mismo paquete dentro del sistema. Esto no obedece a ningún requerimiento especial y podrían haberse realizado en paquetes distintos, pero por simplicidad, y ya que tal cual se encuentran implementados se necesitan entre sí, se prefiere dejar así.

El nodo openni_tracker es distinto, porque si bien es relativamente fácil de instalar, dicha instalación se realiza en forma de diversos archivos, paquetes y librerías esparcidas por el sistema, no ubicadas en una sola carpeta como en el caso delos nodos anteriores.

En la instalación para el correcto uso del paquete gestos y los dos nodos desarrollados en él hay dos principales requisitos. En primer lugar, el uso de catkin como herramienta de instalación y compilación, en alternativa a rosbuild, ya que se ha utilizado de base para el desarrollo de todo, incluyendo

el de los robots sobre los que va incorporado el sistema de visión artificial. En segundo lugar, las dependencias del paquete gestos, que en este caso son los siguientes paquetes:

- *Roscpp* y *rospy*: son los paquetes necesarios para realizar la programación en C++ y en Python respectivamente, de los diferentes nodos del sistema. No tienen una función específica, simplemente son la base para que las funciones de C++ y Python funcionen correctamente.

- *Std_msgs:* es indispensable para el funcionamiento de los mensajes que se intercambian entre los diferentes nodos que conforman el sistema. Sin ello, estos mensajes no estarían especificados.

- *Tf:* es la dependencia que hace posible que se reconozcan los tf creados y comunicados a través de los sistemas de reconocimiento.

Por último, para abordar la ejecución de los sistemas, se ha de recordar que en el capítulo cuatro, correspondiente a ROS, se explicó la utilidad de los *launchers* (archivos lanzadores), que pueden ejecutar un conjunto de nodos sin que sea necesario ejecutar varios terminales con cada uno de los nodos. El número de ellos puede ser muy variado y permitir implementar dentro de un mismo paquete diferentes configuraciones y funcionalidades, por lo que las posibilidades de funcionamiento vistas en el apartado anterior, podrían ser ejecutadas con diferentes ejecutables.

Además, también es útil tener varios *launchers* según el robot en el que se quiera realizar la implementación final, para lanzar junto al reconocimiento desarrollado en el presente trabajo, los nodos encargados del funcionamiento del robot específico, ya sea para pruebas o para experimentación final.

En el desarrollo del proyecto se han realizado y utilizado dos principales ejecutables:

- Gestos_sacarino.launch: que lanza los nodos que configuran y posibilitan el movimiento de sacarino, además de los tres sistemas desarrollados.

- Gestos_edubot.launch: que idénticamente, lanzan los nodos referidos al movimiento del robot Edubot y de nuevo también los tres sistemas desarrollados a lo largo de este proyecto.

Capítulo 7:

Resultados y validación experimental

En los dos anteriores capítulos se han diseñado e implementado las distintas funciones de visión artificial, pero es necesario realizar una serie de pruebas y experimentos destinados a comprobar el grado de consecución de los objetivos marcados en el inicio del proyecto.

7.1 Diseño de experimentos.

Antes de empezar a desarrollar y ver en detalle las pruebas realizadas y sus resultados, conviene conocer algunas bases de las mismas, como el estado de las plataformas robóticas de pruebas utilizadas o el entorno en el que se han realizado dichas pruebas.

Ya en el capítulo quinto se analizaron numerosos detalles sobre los robots que pueden hacer uso del sistema de visión artificial y algunos de sus requisitos. Aunque sea un sistema de visión artificial desarrollado pensando en un objetivo e implementación específica, es completamente extrapolable a cualquier otro robot de tipo social que use el sistema operativo ROS y una arquitectura mínimamente semejante a la trabajada.

En concreto, podemos hablar de tres tipos de pruebas, cada una de ellas realizada sobre un sistema distinto:

- **Turtlesim** (*figura 7.1):* como se comentó en el capítulo correspondiente a ROS, pero existe un paquete que consta de una ventana en 2D por la que se mueven unas tortugas dibujadas siguiendo un determinado patrón. Esto se ha utilizado principalmente para trabajar con Kinect y el resto de sistemas sin necesidad de disponer de un robot físico en todo momento. Obviamente no ha tenido ninguna validez en las pruebas finales con objeto de ratificar los objetivos, pero ha sido de mucha ayuda en pruebas determinadas para comprobar que los gestos daban lugar a cambios en las acciones de un sistema ajeno, en este caso una de las tortugas.

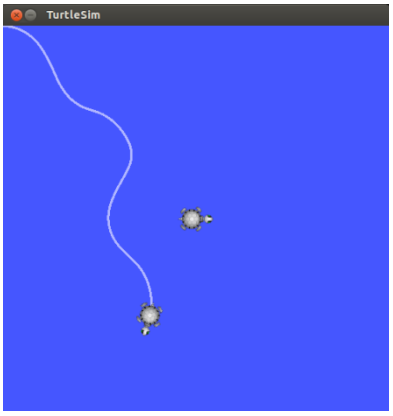

Figura 7.1. Turtlesim.

- **Robot Edubot** (*figura 7.2*): es uno de los robots principales para los ensayos reales, y el que se ha utilizado de base para ajustar el control del movimiento de forma adecuada. Sin embargo, sus limitaciones sobre todo en altura han imposibilitado su uso como la plataforma final de pruebas. Se ha descrito con detalle en el capítulo cinco de esta memoria.

Figura 7.2. Edubot.

- **Robot Sacarino** (*figura 7.3*): es el robot más parecido al tipo de robot que debería implementar la cámara y los programas elaborados. Tiene la

altura suficiente y el sistema de movimiento es muy parecido al del robot Edubot, al menos en cuanto al software. Es el robot utilizado para las pruebas finales con objeto de validar los objetivos principales del proyecto. También ha sido descrito en el capítulo quinto de esta memoria.

Figura 7.3. Sacarino.

Para utilizar los diferentes robots, los programas han de ser adaptados a ellos, y además se han creado diferentes ejecutables que lanzan unos nodos u otros de una manera u otra.

Respecto al lugar de realización de las pruebas de los sistemas, conviene destacar algunos aspectos del entorno que puedan haber influido en los resultados, pudiendo modificar el comportamiento que se puede esperar del robot en el entorno real (que se va a suponer la recepción de un hotel, pero que puede ser cualquier otro de semejantes características)

Figura 7.4. Mapa 2D del entorno de pruebas.

- Los obstáculos: han sido una de las mayores diferencias del entorno de pruebas con respecto al entorno en el que se espera que Sacarino 2 se desenvuelva. En el primer entorno, el lugar estaba lleno de muebles y en general distribuido en forma de pasillos. (como se aprecia en el mapa 2D de la *figura 7.4*), por ello, el espacio para manejarse, sobre todo teniendo en cuenta la distancia que hay que mantener respecto al robot y la problemática de los giros en estas situaciones. Sin embargo, en el entorno real, al ser un espacio amplio sin demasiados obstáculos, o al menos no tantos como en el de pruebas, es de esperar que esta problemática desaparezca.

- Las condiciones de iluminación: en el lugar de pruebas eran muy adecuadas, aunque en principio no es completamente indispensable. Como el sistema de reconocimiento utiliza sobre todo la profundidad (los rayos infrarrojos) para seguir al usuario, es posible que los resultados en un entorno de menos luz como es el de la recepción de un hotel, sean ligeramente inferiores. No obstante, esto es pura especulación.

- Influencia de otros dispositivos: como ya se ha explicado, Kinect utiliza la luz infrarroja como sistema para detectar la profundidad en tres dimensiones, por lo que si se utilizan otros sistemas y dispositivos que utilicen este tipo de señales, puede ser que la superposición de ambas ocasiones problemas en la señal recibida por la cámara. En el entorno de pruebas no había ningún otro dispositivo de este tipo activado, pero si en el entorno real comparte espacio con alguno, es conveniente asegurar que no se obstruyen entre sí.

Por último, a continuación en las imágenes de las *figuras 7.5* (entorno de pruebas) y *7.6* (posible entorno real) se puede ver una comparación entre partes de los dos tipos:

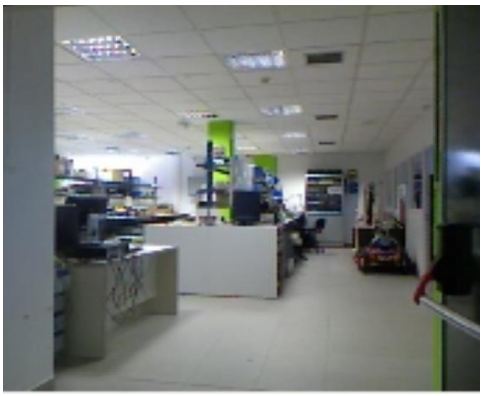

Figura 7.5. Entorno de pruebas.

Figura 7.6. Entorno real.

7.2 Ubicación óptima de Kinect.

Uno de los puntos más importantes en relación al hardware del robot es el lugar de colocación de Kinect. Para esto es necesario no sólo tener en cuenta el funcionamiento óptimo de la plataforma, sino el diseño del propio robot final. Por ello, en conjunción con la persona encargada de modelar el robot Sacarino 2 final, se han estudiado las distintas posibilidades de colocación de Kinect en el robot. Es muy importante esto, puesto que una incomunicación entre personas en el mismo proyecto, aunque con objetivos dispares, puede dar lugar a problemas en las inclusión de funcionalidades en el robot final.

Para empezar, sobre la propia plataforma móvil no era posible colocarlo, puesto que ya se utiliza ese lugar para colocar el láser en dos dimensiones cuyo funcionamiento ya está probado. Así pues, las dos posiciones posibles según el diseño de Sacarino son inmediatamente debajo o encima de la pantalla táctil que incorpora.

Las alturas exactas de esto son, como se puede ver en la *figura 7.7* y sus cotas, 861.50 mm y 1113 mm.

Figura 7.7. Medidas de las alturas de Kinect en Sacarino 2.

Así, teniendo estas dos posiciones, es interesante realizar un estudio de las diversas posibilidades de orientación de Kinect (recordando que tiene un rango de visión de 43,5 grados, una movilidad vertical de 54 grados y tiene un total de 10 inclinaciones diferentes, lo que fácilmente se puede calcular como 5,4 grados de movimiento en cada una de estas posiciones) para determinar la posición más favorable a los intereses de la aplicación. Los diferentes resultados de los límites del rango vertical a diferentes alturas, inclinaciones y distancias se muestran en la siguiente tabla (*7.1*):

		Rangos de alturas (en mm)							
		A 500 mm		A 1000 mm		A 1500 mm		A 2000 mm	
Altura	Inclinación	Pto. Inf.	Pto. Sup.	Pto. Inf.	Pto. Sup.	Pto. Inf.	Pto. Sup.	Pto. Inf.	Pto. Sup.
861 mm	0 (horiz.)	659	1063	457	1265	255	1467	53	1669
	5,4 °	706	1110	551	1359	396	1608	242	1858
	10,8°	753	1157	646	1454	537	1749	431	2047
	-5,4 °	612	1016	363	1171	114	1326	-136	1480
	-10,8°	565	969	268	1076	-27	1185	-325	1291
1113 mm	0 (horiz.)	911	1315	709	1517	507	1719	305	1921
	5,4 °	**958**	**1362**	**803**	**1611**	**648**	**1860**	**494**	**2110**
	10,8°	1005	1409	898	1706	789	2001	683	2299
	-5,4 °	864	1268	615	1423	366	1578	116	1732
	-10,8°	817	1221	520	1328	225	1437	-73	1543

Tabla 7.1. Campo de visión vertical de Kinect.

En base a los datos de la tabla anterior, y teniendo en cuenta que la estatura de una persona común oscila entre 1700 y 1850 mm, su cintura en torno a 900 y 1000 mm, y que cuanto más horizontal esté la cámara mejores resultados se obtendrán, se ha determinado que el lugar óptimo de colocación de Kinect para capturar gestos con las manos es en la parte superior de la pantalla del robot, con una inclinación de 5,4º como se ve en la *figura 7.8*.

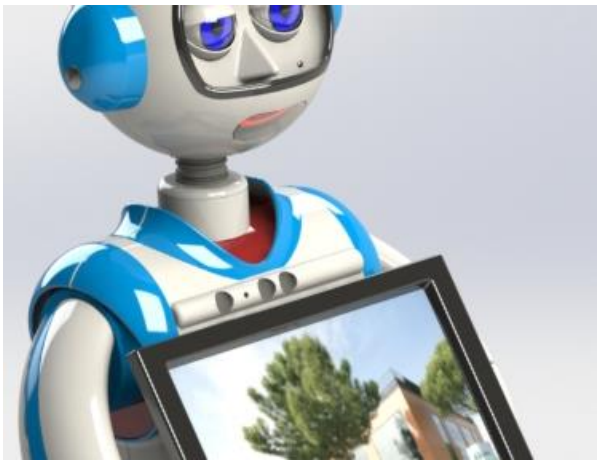

Figura 7.8. Kinect en su implementación final.

No obstante, como las pruebas se han realizado sobre los dos robots previamente mencionados, Edubot y Sacarino 1, y éstos tienen diferentes alturas, los resultados pueden haber quedado ligeramente afectadas, lo que se ha paliado mediante el cambio en orientación según la altura de colocación. En concreto, las pruebas se han realizado de la siguiente manera:

- En Edubot, robot utilizado de manera breve en las primeras pruebas de seguimiento Kinect ha sido colocado en la parte delantera, en el lugar donde

antes estaba colocado el láser 2D. Sin embargo, debido a la baja altura del robot, Kinect debía estar orientado hacia arriba, lo que disminuía su eficacia para el reconocimiento de gestos.

- En Sacarino, robot mucho más parecido al tipo de robot que puede hacer uso del sistema de visión 3D, Kinect ha sido ubicado sobre su pantalla, a una altura de 1250 mm en un punto muy similar al que estará colocado en Sacarino 2.

7.3 Resultados y consecución de objetivos.

Como se determinó en el capítulo de inicio, los objetivos principales del proyecto eran los siguientes:

- Elección de un dispositivo adecuado para la interacción mediante visión artificial, teniendo en cuenta las características hardware y software del robot previamente establecidas.

- Programación del reconocimiento de personas y la interacción con el entorno de acuerdo a los estímulos externos recibidos.

- Realización de todo lo anterior con la mejor relación entre prestaciones y coste del sistema.

Los tres objetivos han sido alcanzados satisfactoriamente. En concreto el primer objetivo se ha resuelto en los capítulos cuarto y quinto. Teniendo en cuenta el robot en desarrollo que hará uso del sistema, Sacarino 2, y el robot utilizado en las pruebas, Edubot, y sus especificaciones, se ha llegado a la conclusión de que el mejor sistema de visión artificial para las funciones a desarrollar es Kinect. Además, también se ha escogido la posición óptima para su correcto funcionamiento, como se ha visto en el apartado anterior.

El tercer objetivo, relacionado con el coste, no se ha visto detalladamente aún, pero se hará en el capítulo posterior. No obstante, tan sólo el precio del dispositivo de visión artificial, Kinect, sensiblemente inferior a otros sistemas similares en el mercado, ya es suficiente para considerar el objetivo cumplido, sin entrar a valorar otros costes.

El objetivo con más consistencia es obviamente el segundo, puesto que es el desarrollo de las funcionalidades del sistema de visión artificial, idea central

de este trabajo. De hecho, este objetivo se desgranó en unas funciones concretas en el capítulo cinco, reproducidas a continuación:

- Reconocimiento de personas.
- Reconocimiento de gestos o posición del cuerpo.
- Seguimiento del robot a un usuario determinado y modificación del movimiento del robot de acuerdo a los gestos del mismo.

En el capítulo anterior se ha explicado el funcionamiento de cada uno de estos sistemas, pero es importante analizar los resultados de los mismos en su conjunto. Al lanzar el ejecutable con el sistema completo, se lanzan los 3 nodos principales desarrollados: el detector de personas, el de reconocimiento de gestos y el que se encarga de dar las órdenes de movimiento al robot, todos por este orden.

En este momento, el primero de ellos queda a la espera de detectar una persona delante de la cámara. Como es de esperar que en la implementación final el usuario inicie este modo de funcionamiento desde la pantalla táctil delantera, el usuario correspondiente será detectado casi de inmediato. No obstante, hay algunas restricciones a este funcionamiento, como la necesidad de que el usuario al que se espera detectar no esté completamente estático delante de la pantalla. Como ya se ha especificado, no es necesario adoptar ninguna pose en especial delante de la pantalla para que comience el rastreo, pero sí es necesario algún movimiento natural, como un paso adelante o atrás o un ligero movimiento del cuerpo en el sitio, para que la calibración completa tenga lugar. En la práctica, esto no tiene relevancia ni ocasiona problema alguno puesto que la sola llegada de una persona al especio delantero del robot es suficiente para iniciar la detección.

En cualquier caso, las pruebas con la calibración y detección de personas, teniendo en cuenta que éstas se han realizado en un entorno con más personas, pero con el usuario principal en primer plano de la cámara (*figura 7.9*), han resultado plenamente satisfactorias, por lo que el primero de los objetivos, el reconocimiento de personas, queda validado.

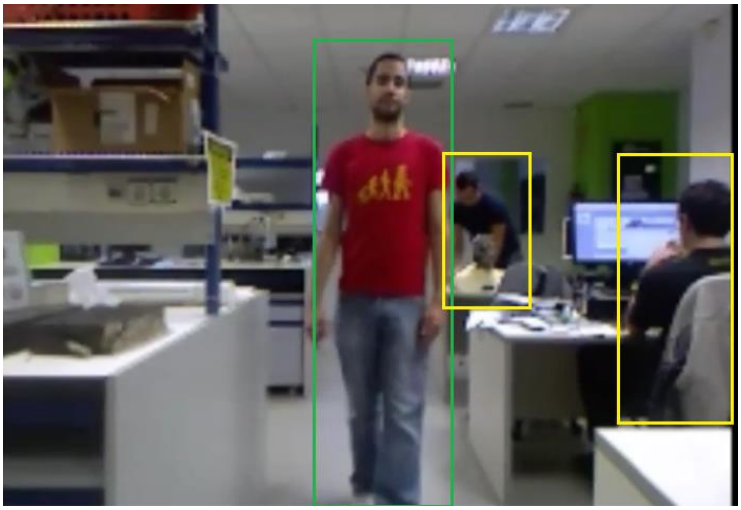

Figura 7.9. Reconocimiento de usuario principal y secundarios.

Una vez reconocido el usuario, los otros dos nodos, que se encuentran a la espera, pueden comenzar a funcionar. Aquí se va a validar el funcionamiento de la programación tal cual está realizada, a pesar de que algunos pequeños cambios puedan cambiar por completo la forma de funcionar del sistema.

En relación a estos nodos, los problemas que pueden aparecer son los asociados a las limitaciones de programación. Por ejemplo, si hay dos o más usuarios delante de la pantalla, es imposible predecir cuál de ellos será el que sea detectado primero y por tanto pueda dar órdenes. Aquí se sucede una de las posibles limitaciones: el sistema no puede adaptarse a nuevos usuarios que den órdenes, se limita a ignorarles. Esto es inevitablemente así porque de no hacerlo, el sistema no tendría forma de ignorar a los usuarios que se interpongan entre el usuario principal y la cámara, y los errores en entornos con varias personas serían numerosos, por lo que en realidad se puede considerar como una ventaja.

Unido a esto, otro resultado importante y que demuestra la robustez del sistema es su capacidad de no cambiar su comportamiento y detectar los gestos y seguir correctamente a un usuario concreto (una vez que ya se ha reconocido como usuario principal) ignorando otros usuarios que se encuentren dentro del campo de visión de la cámara. Esto ya se ha probado y confirmado en las pruebas de comportamiento, obteniendo una fiabilidad muy alta.

Otras pruebas realizadas confirman que, a pesar de que el seguimiento se realiza a cierta distancia y el robot aumenta su velocidad cuanto más lejos esté el usuario, si el usuario va demasiado rápido se puede alejar del robot lo

suficiente como para que éste pierda la referencia de posición del mismo. En ese momento el robot se para, esperando a que vuelva la referencia, y si lo hace en el tiempo suficiente continua moviéndose como estaba establecido. En la práctica, este tiempo es de aproximadamente dos segundos, pasados los cuales es más que probable que el robot no reconozca al usuario inicial y lo trate como a uno nuevo, sin posibilidad de obedecer sus órdenes.

A los méritos anteriores hay que añadir la indiferencia del sistema ante que el usuario esté de frente o de espaldas. Obviamente algunos de los gestos son imposibles de realizar y detectar de espaldas, pero el robot no tiene en cuenta la orientación del cuerpo lo cual al fin y al cabo resulta ventajoso por ampliar la versatilidad del sistema para no incomodar al usuario, según se ha demostrado en las pruebas y se puede ver en las siguientes figuras, según la vista del robot (*7.10)* y una vista externa (*7.11*):

Figura 7.10. Vista trasera desde Kinect.

Figura 7.11. Vista trasera externa.

Las pruebas realizadas han sido con los gestos para activar el seguimiento del robot al usuario, en este caso, mano acercándose (*figuras 7.12 y 7.13*).

Figura 7.12. Vista acercamiento desde Kinect.

Figura 7.13. Vista acercamiento externa.

Y la orden de parada con el usuario delante de la cámara, mediante el gesto de stop (*figuras 7.14 y 7.15*).

Figura 7.14. Vista stop desde Kinect.

Figura 7.15. Vista stop externa.

Con todo esto, queda validado también el comportamiento de los sistemas de reconocimiento de gestos y de seguimiento, con una gran robustez en su implementación.

Sin embargo, la precisión de los gestos con la cámara en movimiento podía dar lugar a errores y malfuncionamientos del sistema, por lo que se han tenido que modificar algunos comportamientos con objeto de paliar esto. De

todas formas, a continuación a continuación se analizan con detalle algunos de estos aspectos y pruebas de los sistemas.

7.4 Pruebas y verificación de las condiciones de operación.

Después de conocer los resultados generales del funcionamiento de los programas y comprobar cómo se han alcanzado los objetivos relativos a las funciones que se deseaban implementar, conviene desgranar los resultados de funcionamiento de cada uno de los sistemas para determinar sus límites y puntos fuertes, a fin de salvarlas en la implementación actual o abordarlas en futuros desarrollos y mejoras.

7.4.1 Sistema de reconocimiento de personas.

Entrando en los detalles sobre el sistema de reconocimiento y rastreo de la posición de personas, y yendo más allá del funcionamiento general validado en el apartado previo, es interesante observar determinadas formas especiales de funcionar y limitaciones intrínsecas al sistema de reconocimiento, separado del resto de nodos y sistemas.

Algunas de estas observaciones sobre el funcionamiento del sistema de reconocimiento de personas son:

- Número máximo de usuarios: las especificaciones del nodo rastreador especifican que el número máximo de usuarios de los que es posible almacenar sus datos reconocidos son 15. Sin embargo, es muy improbable que en el rango de visión que puede detectar la cámara (que al final queda reducido a una superficie de apenas 4 o 5 metros cuadrados) se junte este número de usuarios. En las pruebas el máximo número de usuarios que se han llegado a poner delante de la cámara ha sido 4, y en ningún momento el sistema ha dado muestras de ralentizarse ni dar errores.

- Altura para el reconocimiento: de nuevo, ha sido difícil encontrar personas para las pruebas que se salgan de la norma para comprobar los límites del sistema. El caso de la altura es relevante en tanto que el sistema espera ser usado tanto por adultos como posiblemente por niños. Sin embargo, sí se han hecho pruebas con un niño de 135 cm de altura y, si bien la calibración ha tardado un poco más que para usuarios adultos, ha sido

igual de efectiva. Aquí, en este caso, la limitación puede venir por el lugar en el que se sitúa la cámara, porque si el rango vertical de la misma es capaz de detectar las extremidades superiores de un adulto, es posible que quede fuera del alcance de los niños.

- Falsos positivos: en ocasiones, el sistema detecta como un posible nuevo usuario algún elemento del entorno que claramente no lo es, empezando así una calibración que normalmente no llega a nada. Esto, sin embargo, es una pérdida de tiempo y recursos que podrían y deberían evitarse y que conforman uno de los mayores puntos débiles del sistema. La mayor parte de estos falsos positivos están ocasionados por objetos de gran tamaño del entorno que se mueven, como sucede por ejemplo en la *figura 7.16*, donde la cámara y su trípode son detectados a mayores como un posible nuevo usuario aparte de los dos ya existentes, pero nunca llegan a convertirse en un usuario reconocido y son descartados rápidamente.

Figura 7.16. Falso positivo en la cámara.

- Repetición de usuarios: especialmente con el sistema en movimiento y girando, puede ocurrir que un usuario deje de ser detectado como el usuario que conformaba y sea detectado como un nuevo usuario, y por lo tanto pierda los derechos de comandancia sobre el robot. No es algo que ocurra frecuentemente, sólo de manera ocasional, pero debe ser igualmente tenido en cuenta.

- Tracking inclinado: en la línea de lo apuntado previamente, hay que recalcar la importancia de colocar Kinect enfocando lo más horizontal posible (como por ejemplo en la *figura 7.17*, porque de lo contrario el rastreo de la posición de las articulaciones de los usuarios puede quedar inclinado, y por lo tanto desplazado. En la *figura 7.18* se ve un ejemplo de esto y en la *tabla 7.2*

se observan los valores de coordenadas de alguna de las posiciones, pudiendo comprobar que algunos de ellos no son exactamente lo que cabría esperar.

Figura 7.17. Kinect en posición vertical.

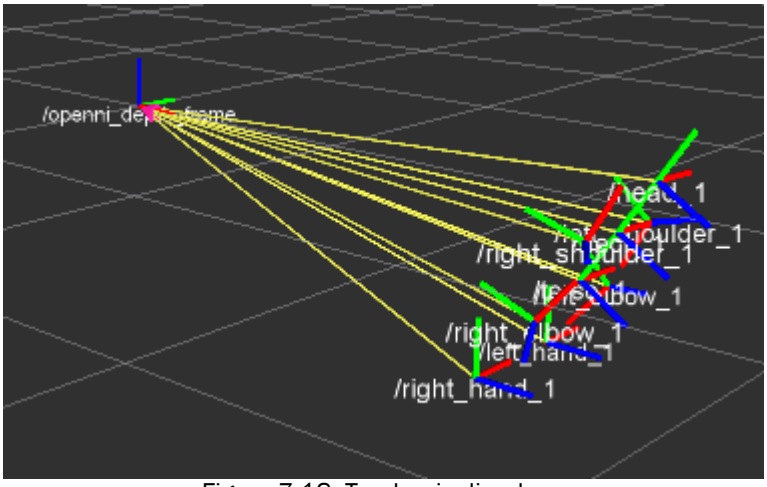

Figura 7.18. Tracker inclinado.

Postura	Coord	Cabeza	Cuello	Torso	Codo der	Mano der	Codo izq	Mano izq
Normal	X	2,16849	2,07558	1,97515	1,97579	1,75885	1,97243	1,76864
	Y	0,05261	0,04926	0,04612	-0,20805	-0,21566	0,29485	0,29582
	Z	0,14749	-0,04444	-0,23203	-0,30368	-0,51953	-0,30783	-0,53354
Mano izq stop	X	2,20923	2,11241	2,01614	2,05986	1,88776	1,74626	1,54508
	Y	-0,01154	-0,00552	0,00464	-0,21978	-0,25297	0,09800	0,03036
	Z	0,03184	-0,13962	-0,31011	-0,47296	-0,65850	-0,02018	0,11284
Brazos en alto	X	2,20109	2,10581	2,00203	2,30394	2,49361	2,32572	2,54689
	Y	0,02359	0,01882	0,01509	-0,14711	-0,07772	0,18273	0,13154
	Z	0,15554	-0,02021	-0,18797	0,30779	0,54486	0,30357	0,52735

Tabla 7.2. Coordenadas erróneas con el tracker torcido.

7.4.2 Sistema de reconocimiento gestual.

Respecto al sistema de reconocimiento de gestos, ya se ha dicho que hay diferencias respecto a si se realiza estáticamente o en movimiento, siendo sensiblemente mejor el reconocimiento de gestos cuando la cámara se encuentra estacionaria. Pero aunque todos los gestos que están en la implementación funcionan correctamente, hay limitaciones a la posibilidad de añadir nuevos gestos.

Por ejemplo, los gestos que cruzan partes del cuerpo, como poner la mano derecha a la izquierda y la izquierda a la derecha deben ser descartados,

puesto que el sistema no tiene la capacidad en la mayoría de ocasiones de distinguir ambos elementos y no es reconocido el gesto como tal.

También deben ser descartados los gestos en movimiento que requieren un movimiento extremadamente amplio en horizontal o sobre todo en vertical, al no existir fiabilidad suficiente de que la distancia del usuario a la cámara sea suficiente para que ésta lo capte completamente (como ocurre en el caso de la *figura 7.19*) sin ser a su vez una distancia demasiado elevada como para que la cámara deje de percibir el reconocimiento del usuario

Figura 7.19. Manos fuera de rango.

Pero los mayores problemas del reconocimiento de gestos son a causa del movimiento de la cámara cuando no se encuentra estática. En estos momentos tiene el trabajo de seguir reconociendo al usuario principal y sus movimientos, pero como la cámara se mueve hacia el usuario, aunque esté parado, lo detecta en movimiento, lo que puede dar lugar a falsos positivos en los gestos o a la no detección de gestos que se supone deberían ser estáticos. Por eso en la implementación final se han simplificado los gestos, con objeto de no ocasionar problemas.

7.4.3 Sistema de seguimiento.

Por último, el sistema de seguimiento por sí mismo no tiene muchos problemas que aclarar ya que realiza su cometido a la perfección, pero sí se puede entrar en detalle sobre el comportamiento en movimiento del robot.

Como en la práctica se han hecho pruebas con dos robots diferentes y la implementación final se debería realizar en un tercero, es evidente que hay que cambiar algunas partes de la programación para adaptar el funcionamiento. Esto es lo que se tuvo que hacer para ejecutar las pruebas pertinentes con el Sacarino original, ya que viniendo del robot Edubot, el

sistema de movimiento presentaba varios cambios, como se reflejó en el capítulo correspondiente a los robots.

Sin entrar en detalles de la programación, otro resultado a tener en cuenta es el de la velocidad del robot y la necesidad de un control para cuando ésta es elevada. En concreto, si se desea una velocidad más alta de la ya establecida, se hace necesario añadir un control de tipo PID o semejante porque la velocidad angular del robot hace que cualquier desviación pequeña de la trayectoria ocasione una oscilación que puede terminar en la inestabilidad completa del sistema.

Por último, en relación al movimiento del robot, se ha explicado en el capítulo anterior cómo el sistema tiene un mecanismo de seguridad de forma que cuando no se ha detectado un usuario o se ha perdido su rastreo, el robot pone a 0 todas sus velocidades y se para. Sin embargo, en la práctica esto no sucede exactamente así y durante el tiempo que tarda en reaccionar el sistema a la pérdida de la posición de un usuario sigue moviéndose, ocasionando que por ejemplo si el usuario se sale del rango de visión de Kinect por un lateral, el sistema continúe dando vueltas hasta que da por perdido el reconocimiento.

No obstante, esto constituye una ventaja, si por ejemplo el sistema no es capaz de seguir a una persona por la elevada velocidad de ésta, continuará el movimiento tratando de recuperar el tracking de esa persona, y si lo consigue en un tiempo lo suficientemente bajo, el sistema continuará funcionando con normalidad como si nada hubiera sucedido.

Capítulo 8:

Estudio económico

En este capítulo se van a detallar los diferentes costes de todo el proyecto, para poder determinar la viabilidad del mismo y si de verdad es realmente ventajoso económicamente respecto a otros trabajos que podrían utilizarse y desarrollarse.

Para ello, se estudiarán los diferentes costes del proyecto, tanto directos como indirectos, incluyendo mano de obra, sistemas hardware y software empleados y otros costes a menudo olvidados pero que es necesario contabilizar, como gastos de desplazamiento o electricidad.

8.1 Recursos empleados.

En todo proyecto tecnológico hay una serie de recursos tanto físicos como software que son utilizados en las diferentes etapas del mismo. Aquí se incluyen los ordenadores, sus sistemas operativos o incluso el material de oficina, por ejemplo.

Una gran ventaja en algunos de los últimos desarrollos tecnológicos es la posibilidad de utilizar hardware y software abierto o también llamado Open Source. Este tipo de sistemas reducen enormemente los costes de un

proyecto al proveer los mismos servicios que sus equivalentes con licencias, pero con coste cero.

En cualquier caso, la totalidad del proyecto ha abarcado los siguientes materiales y recursos, independientemente de si añadirán después un determinado coste o no.

- Sistemas operativos: Windows 7 Professional, Ubuntu y ROS (Robotic Operating System).

- Otros programas: paquete ofimático Microsoft Office 2010.

- Hardware: ordenador portátil Acer Aspire S3.

- Microsoft Kinect: es el recurso principal que diferencia este proyecto de otros del mismo tipo, ya que es un sistema de visión artificial muy versátil y de relativamente bajo precio. Además, posibilita el uso de software abierto, lo que también posibilita un ahorro notable en otros costes.

- Material ofimático: libros de consulta y otros consumibles (folios A4, bolígrafos, lapiceros…).

8.2 Costes directos.

Entre los costes directos se incluyen aquellos imputables directamente al desarrollo del presente proyecto, como puede ser la mano de obra utilizada, los materiales directamente empleados en el sistema desarrollado o la amortización de equipos y programas en su uso en el trabajo como parte de su vida útil total.

8.2.1 Costes del personal.

El proyecto desarrollado ha sido llevado a cabo por un ingeniero, que ha sido encargado de las investigaciones previas, el diseño e implementación de todo el sistema de visión artificial.

También otro ingeniero ha prestado ayuda en algunos apartados y momentos puntuales del desarrollo, por lo que también hay que contabilizarlo, a pesar de ser ayuda externa.

Para contabilizar estos costes, en primer lugar hay que tener en cuenta el salario, tanto bruto anual como las cotizaciones a la Seguridad Social (35% del sueldo bruto) y los días que este salario representa.

Lo primero está representado en la *tabla 8.1*:

COSTE ANUAL	
Sueldo bruto más incentivos	35.000,00 €
Seguridad Social (35% sueldo bruto)	12.250,00 €
Coste total	**47.250,00 €**

Tabla 8.1. Salarios.

En segundo lugar, los días de trabajo, se representan en la *tabla 8.2*:

DÍAS EFECTIVOS POR AÑO	
Año medio	365,25 días
Sábados y Domingos	-104,36 días
Días de vacaciones efectivos	-20,00 días
Días festivos reconocidos	-15,00 días
Días perdidos estimados	-5,00 días
Total días efectivos estimados	**220,89 días**

Tabla 8.2. Días de trabajo.

Con el número de días, y teniendo en cuenta una jornada laboral de 8 horas, el cálculo del **total de horas efectivas de trabajo** en un año es:

$$220{,}89\,\frac{días}{año}\;x\;8\,\frac{horas}{día} = \mathbf{1.767{,}12}\,\frac{\boldsymbol{horas}}{\boldsymbol{año}}$$

Con el sueldo anual anteriormente calculado y estas horas efectivas de trabajo, se puede obtener el coste por hora de un ingeniero:

$$\frac{47.250\,\frac{€}{año}}{1.767{,}12\,\frac{horas}{año}} = \mathbf{26{,}73}\;€/\boldsymbol{hora}$$

En la siguiente tabla (*tabla 8.3*) se muestra la distribución temporal aproximada del trabajo de ambos ingenieros en el presente proyecto, en un desglose de las horas de cada una de las partes de que consta el mismo:

DISTRIBUCIÓN TEMPORAL DE TRABAJO	
Formación y documentación	50 horas
Estudio del problema	50 horas

Desarrollo de la aplicación	100 horas
Puesta a punto del sistema	70 horas
Elaboración de la documentación	150 horas
Total de horas empleadas	**420 horas**

Tabla 8.3. Distribución temporal de trabajo.

Por último, para calcular el coste de mano de obra imputable al proyecto, es decir, el **coste personal directo**, se multiplican las horas empleadas por los ingenieros por el coste por hora calculado anteriormente, obteniéndose:

$$420\ horas\ x\ 26{,}73\ \frac{€}{hora} = \mathbf{11.226{,}60}\ €$$

COSTE PERSONAL DIRECTO	**11.226,60 €**

8.2.2 Costes de amortización de equipos y programas

Aquí se incluyen los costes del uso de los diferentes equipos y programas utilizados que tienen una vida útil mayor a la del presente trabajo. Aquí se excluye el uso de sistemas operativos y programas de tipo abierto, como Ubuntu, ROS y derivados, debido a que

Para ello, se calcula su coste total y su tiempo de amortización, que es la vida útil estimada del correspondiente material. A continuación, a dicho producto se le aplica un factor dependiendo del número de años que se haya considerado como tiempo de amortización:

- Ordenador Acer Aspire S3: vida útil, 4 años. Factor de corrección para la amortización, 0,25.

- Microsoft Windows 7 y Microsoft Office 2010: vida útil, 3 años, ya que es el tiempo en el que Microsoft actualiza sus versiones aproximadamente. Factor de corrección para la amortización, 0.33.

Aplicando la amortización al precio total de los productos, queda la siguiente *tabla 8.4*:

MATERIAL	IMPORTE	FACTOR DE AMORTIZACIÓN	AMORTIZACIÓN ANUAL
Microsoft Windows 7	137,00 €	0,33	45,21 €
Microsoft Office 2010	229,00 €	0,33	75,57 €

Ordenador Acer Aspire S3	650,00 €	0,25	162,50 €
Total material	**1016,00 €**		**283,28 €**

Tabla 8.4. Amortización de material.

El coste final por hora de utilización del material es calculado mediante la división de la amortización anual entre el número de horas de uso en dichos equipos, es decir, el número de horas efectivas de trabajo en un año:

$$\frac{283{,}28\,\frac{€}{año}}{1.767{,}12\,\frac{horas}{año}} \simeq \mathbf{0{,}17}\ €/\boldsymbol{hora}$$

Por último, para averiguar el coste real de amortización del material se multiplica el coste por hora por el número total de horas que se han utilizado los equipos y programas mencionados, en todas las etapas de desarrollo del proyecto:

$$420\ horas\ x\ 0{,}17\,\frac{€}{hora} = \mathbf{71{,}40}\ €$$

COSTE DE AMORTIZACIÓN DE MATERIAL DE OFICINA	**71,40 €**

8.2.3 Costes derivados de otros materiales

En estos costes se incluyen los costes de todo tipo de consumibles utilizados en la elaboración del proyecto y sus documentos a lo largo de toda la fase del desarrollo, como por ejemplo papel, tinta, coste de fotocopias externas, almacenamiento de documentos y programas, etc.

También se ha incluido aquí el coste de la cámara de visión artificial Kinect utilizada en todo el desarrollo del proyecto y que se toma como consumible al ser ésta la función para la que va a estar destinada.

En la *tabla 8.5* se realiza una estimación de algunos de estos costes.

MATERIAL	IMPORTE
Microsoft Kinect	150,00 €
Fotocopias externas	20,00 €
Almacenamiento	10,00 €
Otro material de oficina	20,00 €

Total material consumible	**200,00 €**

Tabla 8.5. Coste del material consumible.

COSTE DE MATERIALES CONSUMIBLES	200,00 €

8.2.4 Costes directos totales

Los costes directos totales, que son la suma de los costes directos anteriores, es decir, personal, amortización de equipos y otros materiales, son:

*11.226,60 € + 71,40€ + 200,00 € = **11.498,00 €***

COSTES DIRECTOS	**11.498,00 €**

8.3 Costes indirectos

Los costes indirectos son aquellos gastos producidos por la actividad asociada al proyecto, pero que no se pueden imputar directamente al mismo exclusivamente porque pueden estar involucrados en otros procesos o proyectos.

En la *tabla 8.6* se desarrollan estos gastos:

COSTES INDIRECTOS PARCIALES	
Dirección y servicios administrativos	150,00 €
Consumo de electricidad	100,00 €
Consumo de desplazamiento	50,00 €
Total gastos indirectos	**300,00 €**

Tabla 8.6. Costes indirectos.

Por tanto, los costes indirectos totales ascienden a:

COSTES INDIRECTOS	300,00 €

8.4 Costes totales

Los costes totales son el resultado de sumar los gastos directos e indirectos, siendo el montante total para este proyecto:

COSTES TOTALES	
Costes directos	11.498,00 €
Costes indirectos	300,00 €
Coste total del proyecto	**11.798,00 €**

Tabla 8.7. Costes totales

En conclusión, el coste total del proyecto asciende a un total de:

COSTES TOTALES DEL PROYECTO	11.798,00 €

Capítulo 9:

Conclusión y futuros desarrollos

Llegados al final, es importante hacer una recapitulación del proyecto desarrollado, viendo los objetivos alcanzados, lo que se ha conseguido por el camino y algunas líneas de investigación que podrían aplicarse en el futuro sobre la base del trabajo realizado.

Como ya se ha discutido a lo largo de la memoria, se puede determinar que los objetivos principales planteados en el inicio se han alcanzado satisfactoriamente gracias a un adecuado análisis y formación previos para abordar correctamente el desarrollo.

El dispositivo de visión artificial elegido, Kinect, cumple con los requisitos necesarios para la realización de las funciones que se esperan de un sistema de visión 3D, e incluso posibilita el desarrollo de futuras líneas de investigación.

Por otro lado, el diseño e implementación de las funciones de visión artificial ha cumplido con las expectativas, obteniendo unos resultados plenamente satisfactorios en las todas pruebas realizadas, incluyendo la detección de personas, él reconocimiento de hasta una decena de posibles gestos distintos y la capacidad de hacer al robot seguir a un determinado usuario cuando este lo ordene.

Respecto a la relación entre prestaciones y coste del sistema desarrollado, se ha podido corroborar que los gastos son relativamente bajos, en especial el más diferenciador dentro del ámbito del trabajo realizado: el sistema de visión artificial. El uso de Kinect otorga no sólo un sistema de visión artificial completamente fiable a bajo coste, sino también la posibilidad de utilizar software *Open Source* a coste nulo, algo que otros dispositivos no serían capaces de soportar.

Todo esto ha supuesto un importante logro y avance en el campo de la robótica y en el de la visión artificial, especialmente en los robots estudiados, al no existir un sistema previo que se adaptara a las condiciones de operación.

El trabajo no ha sido sólo el desarrollo de diferentes programas con unas funciones específicas, sino un estudio completo de los distintos sistemas para conocer en profundidad las necesidades y objetivos concretos a conseguir.

Además de la consecución de estos objetivos, y como parte de lo que debe ser un trabajo de investigación en general y un trabajo de fin de grado en particular, se han adquirido nuevos conocimientos y puesto en práctica otros tantos aprendidos a lo largo de los años previos, en la carrera.

Entre las competencias adquiridas gracias a este trabajo se encuentra como gran exponente ROS, una de las más novedosas infraestructuras de programación de robots que actualmente se encuentra en expansión, por lo que su aprendizaje puede resultar muy útil para la futura vida profesional de cualquier ingeniero que espera dedicarse a la robótica. También se han puesto en práctica y aprendido conocimientos específicos sobre dispositivos y métodos de programación relacionados con la visión artificial, especialmente en tres dimensiones, gracias a la utilización de un sistema sencillo pero muy eficiente como es Kinect.

No obstante, debido a la rapidez con la que avanzan este tipo de tecnologías en la actualidad, siempre hay amplias posibilidades de ampliación y mejoras sobre el sistema desarrollado.

Respecto a las líneas de investigación que podrían abordarse en el futuro sobre la base del trabajo desarrollado, una de ellas podría ser, como se apuntó en el capítulo de análisis y elección de las funciones, la adición de un sistema de mapeado y localización en tres dimensiones (SLAM). El SLAM es una técnica utilizada en robótica por la que los robots son capaces de localizarse en su entorno. En los robots trabajados, y en la mayoría de robots comerciales y de investigación existentes, es común realizar este mapeo automático en dos dimensiones, mediante un láser situado en la parte inferior

del robot. Sin embargo, esto conlleva muchos inconvenientes, como el hecho de que el sistema sea incapaz de detectar obstáculos en alturas diferentes a la establecida (por ejemplo una mesa de pie central).

Por ello podría resultar de interés realizar esto en tres dimensiones con Kinect. La principal limitación reside en el campo de visión del dispositivo, que si bien resulta satisfactorio para tareas de interacción hombre-robot, puede ser insuficiente para el mapeado y la localización en espacios amplios (como por ejemplo el hall de un hotel).

Otra vía de investigación futura consistiría en utilizar un dispositivo Kinect para la detección y el reconocimiento de obstáculos. Para ello se activaría en segundo plano un módulo ROS que, trabajando con las nubes de puntos proporcionadas por Kinect, fuera capaz de reconocer y registrar en tiempo real obstáculos en el camino del robot e informar de ello al sistema de navegación para evitar posibles colisiones. También se podría desarrollar un software de reconocimiento de objetos que, trabajando con las nubes de puntos, fuera capaz de reconocer objetos pequeños en el entorno del robot para su manejo mediante algún actuador tipo brazo robótico.

Por último, para aumentar el rango de visión del sistema 3D, se pueden añadir más Kinects en otras ubicaciones del robot. Sin embargo, para hacer esto es necesario evitar que los puntos infrarrojos que emiten las distintas Kinect no se solapen entre sí, pues esto produciría interferencias entre las ccámaras.

En cualquier caso, la posibilidad de seguir futuras líneas de investigación y desarrollo hace ver que el proyecto expuesto aquí puede ser la base de un amplio ecosistema de aplicaciones y funciones relacionadas con la visión artificial fácilmente incorporables en robots de todo tipo.

Bibliografía y referencias

[1] F. Alhwarin et al *IR Stereo Kinect: Improving Depth Images by Combining Structured Light with IR Stereo*. Aachen University of Applied Sciences, 2013.

[2] C. Blanco. *Desarrollo de un sistema de navegación para un robot móvil.* Universidad de Valladolid, 2014.

[3] Botlr. Aloft Hotels. Disponible y consultado en diciembre de 2014 en: https://www.youtube.com/watch?v=hIpZ8SRK1xo

[4] DARPA SCENICC. Sitio web, consultado en diciembre de 2014: http://www.darpa.mil/Our_Work/DSO/Programs/Soldier_Centric_Imaging_via_Computational_Cameras_%28SCENICC%29.aspx

[5] M. Ferre, R. Aracil and M.A. Sanchez-Uran. *Stereoscopic Human Interfaces - Advanced Telerobotic Applications for Telemanipulation*. IEEE Robotics & Automation Magazine. 2008.

[6] E. Guizo. *How Google's Self-Driving Car Works*. IEEE Spectrum, 2011. Disponible y consultado en diciembre de 2014 en: http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works

[7] D.I.B. Hagebeuker. *A 3d time of flight camera for object detection*. 2007.

[8] P. Henry et al. *Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments*. 12th International Symposium on Experimental Robotics (ISER), 2010.

[9] Y.Y. Huang and M.Y. Chen. *3d object model recovery from 2d images utilizing corner detection*. In System Science and Engineering (ICSSE), IEEE, 2011.

[10] D. P. Huttenlocher. *Comparing images using Hausdorff distance*. Pattern Analysis and Machine Intelligence, IEEE, 2002.

[11] S. Izadi et al. *KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. Microsoft Research Cambridge*, 2011.

[12] K. Lai et al. *A Large-Scale Hierarchical Multi-View RGB-D Object Dataset.* 2010.

[13] Leap Motion. Sitio web, consultado en diciembre de 2014: https://www.leapmotion.com/product

[14] Mantis Visión. Sitio web, consultado en diciembre de 2014: http://www.mv4d.com/

[15] MIT-ros-pkg. Disponible en, consultado en diciembre de 2014: http://wiki.ros.org/mit-ros-pkg/KinectDemos

[16] MOOS. Sitio web, consultado en diciembre de 2014: http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/

[17] S. Morked. *Using the Kinect Sensor for Social Robotics.* Norwegian University of Science and Technology, 2011.

[18] NASA. Sitio web, consultado en diciembre de 2014: http://www.nasa.gov/

[19] Orca. Sitio web, consultado en diciembre de 2014: http://orca-robotics.sourceforge.net/

[20] PAL Robotics. Sitio web, consultado en diciembre de 2014: http://pal-robotics.com/es/

[21] Riegl. Sitio web, consultado en diciembre de 2014: http://www.riegl.com/

[22] Robot AIKO, propiedad de Toshiba. Sitio web, consultado en diciembre de 2014: http://www.toshiba.co.jp/about/press/2014_10/pr0601.htm

[23] Robot ASIMO, propiedad de Honda. Sitio web, consultado en diciembre de 2014: http://newsroom.honda.co.uk/en/corporatepresspackdetail/?id=4152

[24] ROS. Sitio web, consultado en diciembre de 2014: http://www.ros.org/

[25] ROS Wiki. Sitio web, consultado en diciembre de 2014: http://wiki.ros.org/

[26] Sacarino, Cartif. Disponible en web, consultada en diciembre de 2014: http://www.cartif.com/component/k2/item/477-investigadores-de-cartif-desarrollan-un-robot-asistente-para-hoteles.html

[27] A. Saxena, M. Sun, y A.Y. Ng. *3D reconstruction from sparse views using monocular vision*. IEEE 11th International Conference on, IEEE, 2007.

[28] I. Torralba et al. *MIT-CSAIL Database of Objects and Scenes.* Sitio web, consultado en diciembre de 2014: http://web.mit.edu/torralba/www/database.html

[29] Urbi. Sitio web, consultado en diciembre de 2014: http://www.gostai.com/products/urbi/

[30] S. Xiang. *Learning a Mahalanobis distance metric for data clustering and classification*. 2008.

[31] Xtion Pro. Web, consultada en diciembre de 2014: http://www.asus.com/es/Multimedia/Xtion_PRO/

[32] YARP. Sitio web, consultado en diciembre de 2014: http://wiki.icub.org/yarp/

ANEXO:

Hoja de características de Kinect

KINECT™ for Windows®

Name	
Product Name	Kinect™ for Windows®
Product Dimensions	
Length	280mm, 11 in
Width	70mm, 2.8 in
Depth/Height	65mm, 2.6 in
Weight	556.5 g, 1.227 lb
Cable Length	1.524 m, 5 ft
System Requirements	
Interface	Dedicated USB 2.0 Bus
Operating Systems	Requires Windows 7, Windows 8, Windows Embedded 7 or 8
Computer/processor	32-bit (x86) or 64-bit (x64) processor
Memory	2 GB RAM
Imaging Features	
RGB Sensor Technology	CMOS sensor technology
RGB Sensor Resolution	1280 x 1024 pixels @ 15 fps; 640 x 480 pixels at 30 fps
RGB Sensor Imaging Rate	Up to 30 frames per second
RGB Sensor Field of View	Horizontal: 61.5 ° Vertical: 50.0 ° Diagonal: 75.2 °
IR Cut Filter	Pass Band Wavelength: 400-600nm Pass Band Transmittance: > 90% absolute
IR Sensor Resolution	1280 x 1024 pixels @ 15 fps; 640 x 480 pixels at 30 fps
IR Sensor Field of View	Horizontal: 58.0 ° Vertical: 45.1 °
IR Projector Field of View	Horizontal: 57 ° Vertical: 43 °
Operational Range	.4m – 3.5m
USB Bandwidth	
Depth	101 Mbps
RGB	73 Mbps
Audio Out	4.6 Mbps
Motor Control	0.1 Mbps
Total	178.7 Mbps
Audio Features	
Audio Subsystem	4 microphone array
Sample rate and resolution	Each audio channel is sampled at 16 KHz with 24 bits of resolution
Product Feature Performance	
Motor	Tilt Range +/- 30 degrees, Manual pan support +/- 45.1 degrees
Storage Temperature & Humidity	-40 to +60 degC, -40 degF to +140 degF
Operating Temperature & Humidity	+5 to +40 degC, +41 degF to +104 degF

Engineering Specification Number
S011320

Description
Carton, Litholam Retail, Xbox Sensor POM

Carton Face Size Custom
Length : 14.811 in 376 mm
Width : 5.875 in 149 mm

Carton Style
Non-Flap

Spine Size
Other : 4.787 in 122 mm

Carton Closure
Tuck Sealed End Top & Bottom

Joint Type
N/A

Inside Length
14.311 in OR 363 mm

Inside Width
5.6888 in OR 144 mm

Inside Depth
4.662 in OR 118 mm

Outside Length
14.811 in OR 376 mm

Outside Width
5.875 in OR 149 mm

Outside Depth
4.787 in OR 122 mm

Dimensional Tolerance
.031 in (0.8mm) (1/32 in)

Weight
0.450 lb OR 204 grams

KINECT™
for Windows®

Certification Information	
Country of Manufacture	China
ISO 9001 Qualified Manufacturer	Yes
ISO 14001 Qualified Manufacturer	Yes
Restriction on Hazardous Substances	Yes
Agency and Regulatory Marks	ACMA Declaration of Conformity (Australia and New Zealand) CE Declaration of Conformity, Safety and EMC (European Union) WEEE (European Union) ICES-003 report on file (Canada) FCC Declaration of Conformity (USA) VCCI Certificate (Japan) GOST Certificate (Russia) KCC Certificate (Korea) UL and cUL Listed Accessory (USA and Canada) CB Scheme Certificate (International)
Windows Hardware Quality Labs (WHQL)	Certified
SKU and Country List	
L6M-00001	US, Canada, Mexico
L6M-00002	UK, Ireland
L6M-00003	France, Italy, Germany, Spain
L6M-00004	Australia, New Zealand
L6M-00005	Japan
L6M-00007	Brazil
L6M-00008	Denmark, Finland, Norway, Sweden, Russia
L6M-00009	Austria, Belgium, Netherlands, Portugal, Switzerland
L6M-00010	Saudi Arabia, UAE
L6M-00011	South Africa
L6M-00012	Hong Kong, Singapore
L6M-00013	Korea
L6M-00014	Taiwan
L6M-00015	India