Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA INGENIERÍAS INDUSTRIALES

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

SISTEMA DE INTERACCIÓN VISUAL PARA UN ROBOT SOCIAL

Autor: MARIO DOMÍNGUEZ LÓPEZ

Tutor: DR. JAIME GÓMEZ GARCÍA-BERMEJO

25 de Junio de 2017

Agradecimientos

A mis padres Mar, José Luis y hermana Raquel, por su gran apoyo. A mis tutores Jaime y Eduardo, por la oportunidad, guía y ayuda prestada. Agradecer también a mis compañeros de división Samuel, Javier y Roberto, sus consejos y conocimiento del tema. A Dios, por estar siempre a mi lado.

Si quieres ir rápido camina solo, si quieres llegar lejos ve acompañado.

Proverbio Africano

Si crees que puedes, ya estás a medio camino.

Theodore Roosevelt

Resumen

Una de las características principales de los robots sociales es que tengan expresividad, y, fluidez, lo más cercana a la naturaleza humana. En este caso una de las mayores fuentes de expresividad son los rasgos faciales. Por otra parte, estamos acostumbrados a comunicarnos mirando a otras personas a la cara por lo que desarrollar robots con expresividad, capacidad gestual y de seguimiento facial es una de las características buscadas en el desarrollo de robots sociales.

El presente proyecto describe el desarrollo de un sistema de interacción hombre-máquina para un robot móvil social constituido por dos subsistemas o módulos. El primer subsistema se encarga de la detección del interlocutor y su seguimiento facial. Para ello realiza una detección inicial a partir de la información suministrada por el láser de navegación del robot, para posteriormente fijar la atención a partir de la información obtenida con una cámara de rango tipo Kinect. El segundo subsistema de encarga de la generación de gestos del robot. Para ello se ha adoptado el estándar FACS (Facial Action Coding System), el cual permite su utilización en diferentes tipos de robots. La generación de expresiones se realiza mediante la combinación del movimiento controlado de cuello, párpados y ojos, con gestos mostrados en un display luminoso en la boca del robot.

Palabras clave: Interacción Hombre-Máquina, Robot Social, ROS, FACS

Abstract

One of the main characteristics in social robots is its expresivity and fluidity. In this sense, the main source of expresivity are facial features. On the other hand, we are accustomed to interacting with each other by staring at their eyes, and this fact forces us to promote robots with expresivity, gestural ability and facial tracking remains to be a key-factor when developing social robots.

The present project approaches the comprehension and design of an Human-Machine interaction system for a social robot based on two modules or sub-systems. The aim of the first module is the detection of the user and its facial tracking. To achieve this, it does an initial detection from the data collected by the robot's navigation laser, in order to draw user's attention later, with the information obtained by a range-camera type as Kinect. The second sub-system is in charge of the robot's gesture generation. This is managed by adopting FACS (Facial Action Coding System) standard, which can be used within different types of robots. The gesture generation is accomplished by the combination of both, the controlled neck movement, eyebrows and eyes, and the gestures shown in a light-based display, placed in the mouth of the robot.

Keywords: Human-Machine Interaction, Social Robot, ROS, FACS

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

PLANTEAMIENTO

A continuación se presenta la introducción a esta memoria, el planteamiento del trabajo, objetivos y justificación.

1.1. Introducción

El presente proyecto se encuadra dentro del ámbito de la robótica, visión artificial y sistemas de tiempo real, tecnologías que juegan un papel fundamental y emergente en el mundo actual. Dentro de la robótica podemos distinguir, al menos, dos categorías:

- La primera es la robótica industrial: enfocada sensiblemente a procesos de producción y empresa. En la que el robot desempeña tareas repetitivas sustituyendo al operador humano, siempre menos preciso que el propio robot.
- La segunda sería la robótica social o de servicios: que trata de dar soporte a todas aquellas aplicaciones en las que el robot está en contacto con el humano, ayudando en diversas tareas dentro de numerosos sectores, como son el hotelero, hospitalario, en el propio hogar, etc.

Aunque esta división no debe ser considerada estanca, de hecho, las tendencias de la robótica actual van enfocadas a robots colaborativos, interaccionando tanto entre ellos como con los humanos, incluso dentro del ámbito industrial (industria 4.0), son los llamados Cobots.

Por su parte, la visión por computador es una disciplina que ha supuesto una revolución en la robótica, y, aunque esta tecnología no fue muy desarrollada en sus inicios debido a la cantidad de recursos hardware que necesitaba, hoy en día se encuentra en pleno auge. Dota al robot del sentido de la vista, y esto tiene consecuencias directas sobre la mejora de las aplicaciones existentes y futuras, puesto que el robot es capaz de conocer la ubicación de, ya sea un objeto, persona o herramienta en tiempo real.

A su vez, los sistemas de tiempo real se hacen imprescindibles a la hora de poner en contacto las tecnologías mencionadas, planificando las tareas correspondientes en cada momento y asegurando una respuesta óptima en términos de seguridad y comunicaciones. Por tanto, este proyecto está enmarcado dentro de la robótica de servicios, y el entorno real en el que se va a mover el robot hace que sea imprescindible la aplicación de visión artificial.

1.1.1. Justificación

El robot Sacarino II es un robot social desarrollado en el centro tecnológico CARTIF, en colaboración con el centro San Luis (Palencia), que tiene como finalidad convivir en un hogar con personas mayores, por tanto, deberá de ser capaz de desempeñar diversas funciones en el lugar como son: acompañamiento, ofrecimiento de información diversa a los tutores,juegos, actividades, etc. Además, para facilitar la interacción hombre-máquina, éste es de aspecto humanoide y se comporta lo más cercanamente a como lo haría un humano.

Esto supone un reto importante, puesto que va a tratar con personas, el robot debe tener una alta expresividad así como una interacción fluida y natural. Otros trabajos enfocados sobre este mismo tema han sido tenidos en cuenta [8] y han servido como apoyo para la realización de este proyecto [9].

1.2. Objetivos

A continuación se plantean los objetivos generales y específicos tanto para el sistema de visión artificial como para las herramientas de configuración

El objetivo general del presente proyecto es el de lograr una mayor expresividad en el robot Sacarino II, para mejorar la interacción hombre-máquina. De los estándares de comportamiento social se desprende la importancia de mirar a una persona a los ojos cuando se está interactuando con ella, puesto que significa estar activo en la conversación; y por otra parte, la gesticulación y expresividad facial acentúan el acercamiento y lo hacen más humano, reforzando positivamente la interacción.

Dentro de este gran objetivo, se engloban tanto aspectos relacionados con la correcta elección del software (elección de librerías de trabajo) y hardware (correcta selección de elementos sensores y actuadores), para que la interacción con los elementos externos sea lo más fluida posible. Esto resulta en los siguientes objetivos específicos:

- Lograr un movimiento natural de la cabeza del robot tal que sea capaz de mantener la mirada fija en el usuario, es decir, todo aquello que tiene que ver con la localización de usuarios más próximos, detección y seguimiento (*tracking*) facial de estos.

- Una vez iniciada una interacción con un usuario, el robot debe de poder expresar gestos y emociones.

- Debido a la complejidad y cantidad de actuadores que conllevan los objetivos ya comentados, y para la correcta puesta en marcha de todo lo anterior, se ha de desarrollar una interfaz gráfica que sea lo más intuitiva y generalista posible, es decir, que además sea reutilizable en otros robots.

- Elaborar todo lo anterior con el mejor equilibrio entre coste y prestaciones.

Además, como consecuencia de estos objetivos planteados, se derivan una serie de objetivos secundarios, como enmarcar los diferentes sistemas dentro de estándares desarrollados por la comunidad científica y dotar de un desarrollo generalista a los programas desarrollados.

1.3. Estructura de este documento

Una vez definidos los objetivos del presente proyecto, se pretende dar una idea de cómo está estructurado el proyecto, haciendo un recorrido por los diferentes capítulos y temas que aborda, a fin de servir como guía orientativa al lector. La memoria se encuentra dividida en tres bloques bien diferenciados, el primero es un bloque de estado de la tecnología actual, a continuación, en el segundo bloque trata el cuerpo del proyecto, presentando los programas y herramientas desarrollados para alcanzar los objetivos marcados y, finalmente, un bloque dedicado a la validación de objetivos, viabilidad económica y conclusiones.

Bloque	**Unidades**	**Descripción**
Bloque I	Capítulos: 2 al 4	Estado de la tecnología actual y robot de trabajo.
Bloque II	Capítulos: 5 al 7	Cuerpo del proyecto, presentando los programas y herramientas desarrollados para alcanzar los objetivos.
Bloque III	Capítulos: 8 al 10	Validación de objetivos, viabilidad económica y conclusiones.

Tabla 1.1: *Resumen Estructura del Proyecto.*

Durante el segundo capítulo se abordarán aspectos relacionados con la robótica de servicios y se hablará de los robots sociales desarrollados en CARTIF, especialmente del robot en el que se va a implementar el sistema de visión artificial, su mecánica, hardware de control y especificaciones.

En el tercer capítulo se hablará del estado de la tecnología de visión artificial, los diferentes sistemas de visión 3D y la elección del sensor de visión a implantar en el robot. En el cuarto capítulo se exponen las librerías, herramientas, la explicación y el uso del sistema operativo usado en el robot así como del entorno utilizado para la programación.

A partir de aquí, se adentrará en el segundo bloque de la memoria, comenzando por el quinto capítulo, referido principalmente al sistema de seguimiento facial y desarrollo

de los programas para llevarlo a cabo.

Seguidamente, a lo largo del sexto capítulo, se profundizará en el sistema de gestos elegido, los niveles de abstracción en la programación, los mensajes y programas creados con este fin e interfaces utilizados, concepto y justificación.

Para cerrar este segundo bloque, abordaremos el sistema global de interacción así como el diseño de la interfaz gráfica para la calibración de los actuadores del robot.

En este punto se entrará el tercer bloque, que comprende los capítulos 8, 9 y 10. En el octavo capítulo se trata toda aquello que tiene que ver con la validación de los objetivos marcados, resultados de los experimentos realizados y funcionamiento general de la interfaz, así como el análisis de lo servicios finales que ofrece.

En el noveno capítulo se desarrolla una aproximación económica al coste real de la realización del proyecto, con el consiguientes desglose por costes de materiales y recursos.Presupuesto.

Finalmente, el capítulo que cierra esta memoria está destinado a conclusiones, aprendizajes, y futuras líneas de desarrollo e investigación.

Capítulo 2

ROBÓTICA SOCIAL. ROBOT SACARINO II

Este capítulo profundiza en el concepto de robótica de servicios, además de analizar el estado del arte en este momento. Para ello se propone un recorrido a través de la historia con el fin de entender el estado presente de dicha tecnología. Por otro lado, nos centraremos en el robot donde se van a introducir los sistemas desarrollados, ahondando en su construcción a distintos niveles, hardware, software y aplicación.

2.1. Evolución de la robótica

A lo largo de la historia, el humano siempre ha intentado construir máquinas automáticas que le pudieran aliviar su trabajo, haciendo sus días un poco más cómodos, satisfacer su curiosidad, su afán de entender, investigar y aprender, o, simplemente, que les sirvieran como entretenimiento [7].

Fue en Grecia donde se construyeron los primeros mecanismos de funcionamiento automático a los que llamaron autómatas; más adelante, en la Edad Media y en el Renacimiento se siguieron procesando diversos autómatas, entre los que destacan el gallo de Estrasburgo (1230) y el león animado de Leonardo Da Vinci. A lo largo de los siglos XVII y XVIII se crearon mecanismos de mayor complejidad que tenían alguna de

las cualidades de los robots actuales; sirva como ejemplo ejemplo Jacques de Vaucanson (1709-1782) que creó varios autómatas, el más ilustre se trata de un pato mecánico, que bebía, comía, graznaba, chapoteaba en el agua y tomaba su comida como si de un pato real se tratase; estos primeros autómatas fueron destinados fundamentalmente a ser presentados en las ferias y servir de entretenimiento a los estamentos más altos de la sociedad, como la nobleza.

Entre finales del siglo XVIII y principios del XIX se desarrollaron algunas máquinas automáticas para empleo en la industria textil entre las que ya había algún telar en el que haciendo uso de pequeñas tablillas con agujeros se podía elegir el tipo de tela que se iba a tejer, este hito, constituye uno de los primeros precedentes históricos de las máquinas CNC programadas por control numérico. Un poco más tarde que en la industria del tejido, se desarrollan los automatismos en los sectores minero y metalúrgico, es la época de las máquinas de vapor; James Watt (1736-1819) contribuye decisivamente al desarrollo de estas máquinas e inventa el llamado regulador de Watt, que es un regulador de tipo centrifugo de acción proporcional; con él nace el concepto de realimentación y la regulación automática, que es una de las bases de los robots industriales actuales, que entre otros elementos, incorporan diversos sensores y reguladores PID.

La palabra robot se acuñó por primera vez en 1920 en una obra de teatro llamada «Robots Universales Rossum» escrita por el dramaturgo checo Karel Capek, se deriva de la palabra checa robotnik y significa, siervo, servidor o trabajador forzado. No es hasta el siglo XX cuando se empieza a hablar de robots y se produce el crecimiento de estos, que va ligado con el desarrollo de los microprocesadores. En la tabla 2.1 se citan los hitos más importantes.

A partir de finales del siglo XX, se sentarán las bases de la robótica industrial, con el posterior crecimiento de empleo y complejidad de los robots debido al aumento de las prestaciones de los microprocesadores y las posibilidades de la informática. El auge de los robots, además del desarrollo de la electrónica, se debe a la necesidad industrial de fabricar productos con variaciones en función de los gustos y necesidades de los clientes, lo cual ha hecho que las máquinas y dispositivos automáticos de fabricación específicos para fabricar un producto único, solo sean rentables para grandes paquetes de producción.

Fecha	Hito
1954	A partir de esta fecha, el estadounidense George Devol, comienza la construcción de un brazo articulado que realiza una secuencia de movimientos programables por medio de computador; se considera que este brazo es el primer robot industrial.
1956	Devol conoció a Joseph Engelberger y juntos fundaron en 1960 la empresa Unimation dedicada a la fabricación de robots.
1961	Se realizan pruebas de un robot Unimate accionado hidráulicamente, en un proceso de fundición en molde en General Motors.
1968	Kawasaki se une a Unimation y comienza la fabricación y el empleo de robots industriales en Japón. En este año General Motors, emplea baterías de robots en el proceso de fabricación de las carrocerías de los coches
1973	La empresa sueca ASEA, fabrica el primer robot completamente eléctrico, es el tipo de accionamiento que ha acabado imponiéndose, debido a las avances registrados en el control de motores eléctricos.
1974	Se introduce el primer robot industrial en España. También es el año en el que se comienza a usar el lenguaje de programación AL, del que derivarán otros de uso posterior como el VAL (Victor´s Assembly Languaje) de los robots PUMA, implementado en 1975 por Victor Scheinman, que junto a Devol y Engelberger, son pioneros en la robótica industrial.
1978	Comienza a emplearse el robot PUMA (Programmable Universal Machine for Assambly) de Unimatión, que es uno de los modelos que más se ha usado, su diseño de brazo multiarticulado es la base de la mayoría de los robots actuales.
1981	Comienza la comercialización del robot tipo SCARA (Selective Compliance Arm for Robotic Assambly) en Japón.
1981	Se constituye la Federación Internacional de Robótica con sede en Estocolmo.

Tabla 2.1: *Resumen Hitos en la Robótica (de [7]).*

2.2. Robótica social

Si hablamos de robots socialmente interactivos, la interacción social juega un papel esencial en su funcionamiento. Un robot controlado de forma remota no puede ser considerado social, puesto que no es capaz de tomar decisiones de manera autónoma. Es la mera extrapolación de un ser humano. pero esto no quita que para que pueda ser considerado social, tenga que ser completamente autónomo.

También es aceptada una autonomía parcial. Por otro lado, esta autonomía deriva de la capacidad obligada de tener ciertas capacidades, tales como recolectar información de su entorno y estímulos externos, trabajar durante largos periodos de tiempo sin necesitar de una intervención humana, moverse en parte o completamente a través de su dominio de actuación sin necesitar asistencia, y evitar situaciones peligrosas, tanto para humanos como para él mismo, a no ser que esté programado para dichos fines.

El principio de la robótica social se remonta a las décadas de los años cuarenta y cincuenta del siglo XX de la mano de William Grey Walter, aunque los primeros desarrollos fueron realizados durante los primeros años de la década de los noventa por investigadores de Inteligencia Artificial: Kerstin Dautenhahn, Maja Mataric, Cynthia Breazeal, Aude Billard, Yiannis Demiris, y Brian Duffy entre otros. Las características que definen un robot social, según [10] son:

(1) Expresan y/o perciben emociones.

(2) Tienen una comunicación con alto nivel de diálogo.

(3) Sostienen un aprendizaje y reconocimiento de modelos de otros agentes.

(4) Establecen y mantienen relaciones sociales.

(5) Usan acciones de entrada/salida naturales (gestos, pestañeo, etc.)

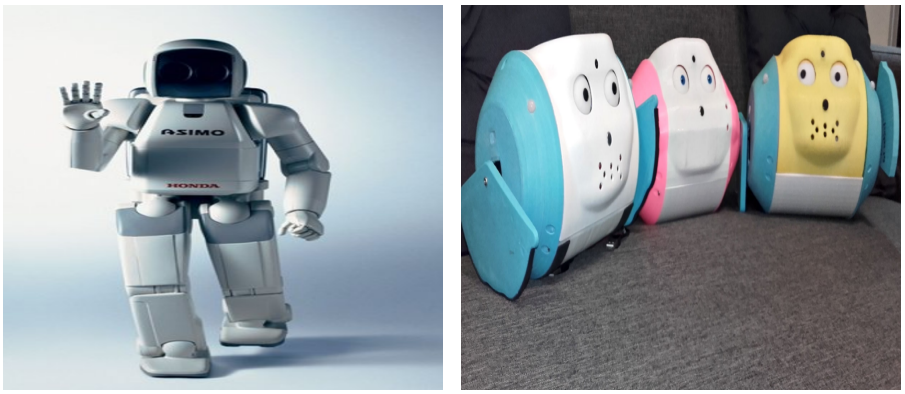
(6) Exhiben una personalidad y carácter particulares y distintivos.

(7) Pueden aprender y desarrollar competencias sociales.

2.3. Morfología e interacción con los humanos

La esencia de la percepción mutua robot-entorno está basada en la naturaleza bidireccional entre dichos sistemas. En la medida en que el robot modifique su campo de acción y éste entorno modificado altere al sistema robot, así será la realización de su estructura.

Los robots sociales no siempre necesitan de un cuerpo físico material tal y como lo podemos entender los humanos. Sirvan como ejemplo los agentes de conversación, que pueden ser la realización del sistema en la misma extensión que un robot con capacidad limitada, T.Sheridan [11]. Resulta evidente que unos robots tienen una realización más compleja que otros, véase si no la diferencia entre *Asimo* (Honda) e *IO*.



(a) Asimo (b) IO

Figura 2.1: Complejidad en los robots sociales

La forma y estructura del robot (Figura 2.6) es importante también en la medida en que va a fijar las expectativas sociales, puesto que la apariencia física es la carta de presentación y predispone la interacción. Un robot con forma animal, va a ser tratado de forma diferente, a priori, que un robot de apariencia humanoide.

En general, la morfología y diseño de un robot puede ocasionar grandes efectos en la expresividad y accesibilidad. Algunos investigadores afirman que las características físicas que muestran los robots sociales deben ir en sintonía con el fin para el que han sido fabricados. Hasta ese punto condiciona el aspecto. Este tipo de realización aparece a menudo en los robots cuyo objetivo es el cuidado de personas, por ejemplo en

asistencia o traslado de pacientes. Así, características como el agarre, asiento, altura, son fundamentales en su diseño para el final desempeño.

En los robots de juguete como IO (fig 2.6 b) el diseño vuelve a jugar un papel clave, puesto que tiende a reflejar sus requerimentos. Los juguetes deben minimizar sus costes de producción para ser competitivos, resultar llamativos y atractivos a niños y ser capaces de modificar su comportamiento a una variedad de situaciones que se puedan dar durante el juego,[12].

Por tanto, resulta evidente que los robots sociales interactivos están empezando a jugar un rol importante en nuestro mundo, trabajando por, para o junto a los humanos. En necesario crear un grado de empatía con los humanos, aprovechando la predisposición de nuestro cerebro, como indica Sherry Turkle, a ser «engañado» por una máquina.

La expresión de emociones , percepción, diálogo y capacidades cognitivas son capacidades necesarias para que un robot sea capaz de simular con el mayor grado de parecido, el comportamiento humano.

2.3.1. El valle inexplicable de Mori

M.Mori definió una regla básica en el campo de la robótica [1]. El valle inexplicable de Mori es una reacción emocional de rechazo contra otros agentes no humanos que intentan asemejarse tanto físicamente como con en su conducta al hombre, por parte de los humanos. Es decir, la hipótesis afirma que cuanto más se parece un robot a un humano tanto en apariencia física como en sus movimientos, más empatía genera en los humanos, pero, llegado un cierto grado de semejanza, se entra en una especia de «valle inexplicable» en el que el robot genera repulsión.

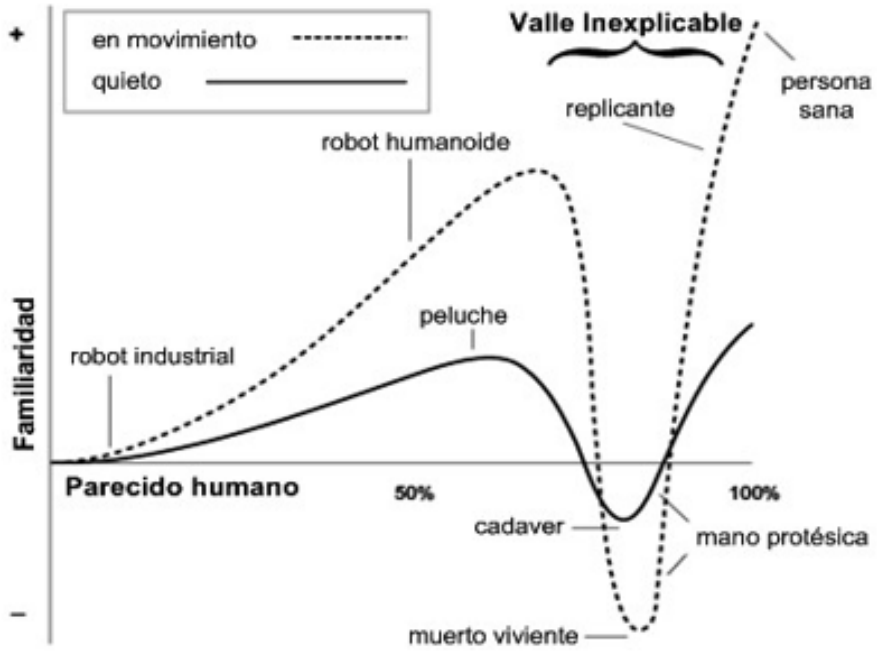


Figura 2.2: El valle inexplicable de Mori (de [1])

Una explicación a este efecto podría ser que, en el caso en que el agente tenga aspecto humanoide, los rasgos «no humanos» destacarán más, creando así una sensación de frialdad y lejanía, con el consiguiente sentimiento de rechazo. Otra, propone que este comportamiento se debe a que algunas acciones y ciertos rasgos de los robots, guarden parecido a los de los enfermos y moribundos, pero que al no tener causa concreta del motivo de este comportamiento, pueda crear en nuestra mente la sensación de riesgo contra nuestra propia integridad, así como poner en una paradoja nuestra lógica inconsciente[10].

Un buen ejemplo de este fenómeno fuera del campo de la robótica es el del personaje de dibujos animados, inspector Gadget. Su comportamiento, rasgos y origen son claramente humanos y además hace resaltar las características más humanas, creando sensación de afinidad.

El experto en robótica David Hanson denota la existencia del valle de Mori como una teoría pseudocientífica. Y argumenta que los diseñadores de robots no deben de

estar influenciados por una teoría sin pruebas científicas [13].

2.4. Sacarino II

Sacarino fue un robot móvil social diseñado por el centro tecnológico CARTIF para trabajar como «botones» en entornos hoteleros. Tras haber sido terminado, pasó un período de prueba trabajando en el hotel NOVOTEL de Valladolid. Del posterior artículo publicado en la revista «Robotics and Autonomous Systems» se desprenden varias conclusiones e ideas que mejoraron la funcionalidad del robot a lo largo de su estancia, como las siguientes [14] :

- El robot debe ser completamente autónomo. Por lo que se ha tenido que desarrollar un sistema de carga de batería que no dependa de ningún personal, esto favoreció enormemente la autonomía del robot.
- La navegación debe ser robusta, pero a a vez suficientemente precisa para situar el robot en el lugar indicado. Varios de los inconvenientes pueden ser evitados mediante el cálculo de nuevas trayectorias para evadir colisiones o situaciones no deseadas. Además, se tuvieron que añadir matrices de datos y ledes infrarojos para, por ejemplo, un mejor guiado en las maniobras de acoplamiento a la estación de carga.
- Como era de esperar, los niños son los que más se acercan a jugar con el robot, agarrándolo de brazos y cabeza, sin mirar lo que dice la pantalla. Esto acarreó algunos problemas sobre todo en el cuello. Por lo que se tuvo que reforzar estas partes, protegiéndolas lo más posible.
- Las normas de estandarización para la interacción humano - robot (ISO/NP ISO 13482) tienen que ser tenidas en cuenta desde el diseño a nivel hardware hasta el de aplicación. hay que prestar especial atención a los sistemas de aislamiento eléctrico tales como baterías, alimentación, velocidad, etc pues la inclusión de estos sistemas pueden penalizar la autonomía del robot. Sirva como ejemplo, cuando algunos usuarios (especialmente jóvenes) pulsan la seta de emergencia sin estar realmente en una situación de peligro, entonces Sacarino queda inoperativo hasta

que un miembro del personal desbloquee el botón de emergencia. Un mecanismo que pueda avisar de forma remota al personal de trabajo podría ser una buena solución a adoptar.

Con el éxito que supuso el funcionamiento y validación de servicios de Sacarino, comenzó entonces la idea de sacar adelante un nuevo robot, con la experiencia aprendida de la primera versión, pero para un fin distinto. Esta vez Sacarino II se destinaría a tratar con personas mayores y/o pacientes con alguna enfermedad.

2.4.1. Morfología y Diseño

El objetivo de Sacarino es el trato con humanos. Esto condiciona el diseño y su morfología, como se ha explicado en el apartado «Morfología e interacción con los humanos». Se ha seguido una línea continuista con el diseño de Sacarino, pero con detalles diferenciadores, por ejemplo, Sacarino II tiene ahora un aspecto mas humano que su predecesor, pues incorpora en su estructura el torso inferior y las piernas.

Base

La plataforma móvil es un módulo robotizado de forma cilíndrica de 50cm de diámetro y altura 40cm, situado en la parte inferior de Sacarino (figura 5-3), dentro del cual se encuentran; un láser tipo SICK, el sistema de energía, el sistema de desplazamiento y la mayor parte de la electrónica de control. Desde el punto de vista mecánico la plataforma es el soporte de todos los sistemas y le otorga al robot la capacidad de movilidad.

La estructura es robusta, diseñada para soportar todo el peso del robot, y está rodeada por un panel lateral, que además de soportar ciertos sensores (bumpers y sonars), oculta y protegen el interior del robot. En el interior se encuentran el sistema de tracción y las baterías, además de distintos dispositivos electrónicos y el PC.

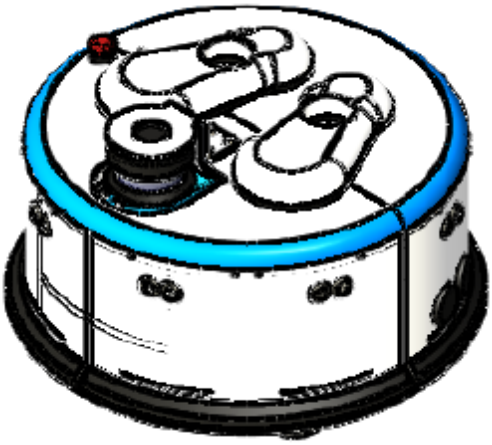


Figura 2.3: Diseño conceptual de la base (de [2])

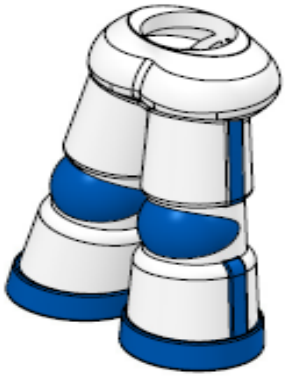
Todo el panel lateral dispone de dispositivos de contacto (bumpers) que transmiten una señal electrónica cuando sufren alguna presión detectando colisiones del robot. Además superiores dispone de un anillo de 8 ultrasonidos. Estos ultrasonidos tienen por misión detener el robot ante un posible obstáculo para evitar colisiones [3].

Cuerpo

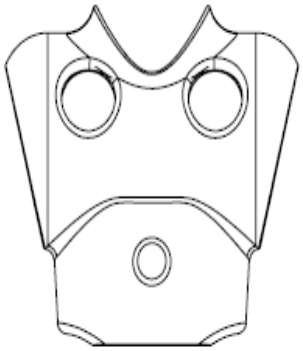
Ya en la parte del cuerpo [2], su estructura interior, lleva unos perfiles de aluminio circulares anclados a la segunda base, que, a la vez de hacer de la sujeción de toda la carcasa, sirven para el paso de los cables desde la base hasta la pantalla y la cabeza.

La carcasa de los pantalones se ha mantenido en todo momento desde la idea inicial, lo que se ha modificado ha sido la camiseta a una forma más simplificada para poder adaptar los componentes adicionales como un juego de altavoces e insertar la pantalla.

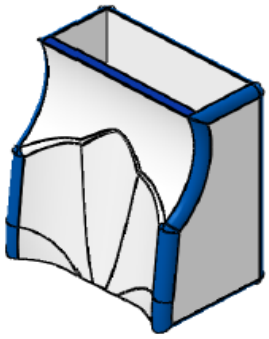
También se añadió una mochila en la parte trasera de la espalda con el fin de añadir la función de transportar objetos o dar información. Esta se sustituirá por un cajón en la base para evitar subir el centro de gravedad.



(a) Piernas



(b) Vista delantera del torso



(c) Vista trasera torso

Figura 2.4: Vistas del cuerpo (de [2])

Cabeza

En cuanto a la cabeza [2], cuya estructura interna se detallará más adelante, se realizaron dos modelos diferentes, algunos más futuristas como los modelos que se han podido ver en el estudio de mercado (como es el caso de Pepper, el robot social de SoftBank Rootics [1]), que, su forma permite la colocación de altavoces en los laterales.

Finalmente, se determinó seguir con la misma línea del prototipo anterior, pues el diseño de la cabeza en versiones anteriores fue recibido con gran éxito.

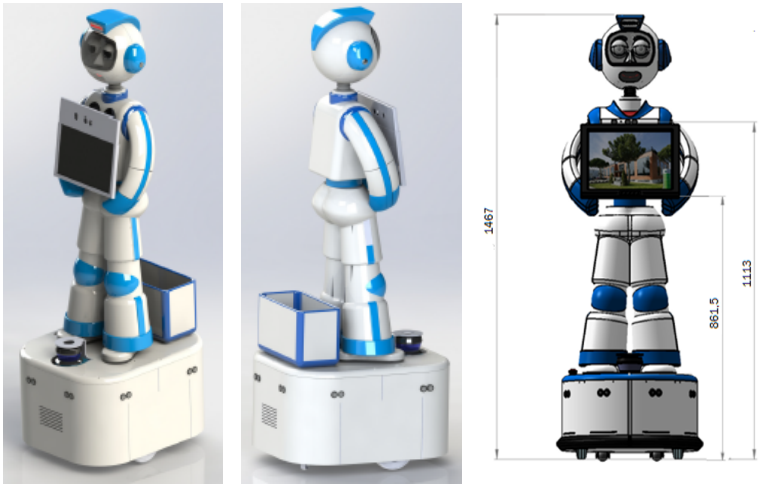
[1] `https://www.softbank.jp/en/robot/`

Figura 2.5: Cabeza

La boca de Sacarino II es una pantalla formada por una matriz de ledes rojos, que se encienden de diferentes maneras para simular diferentes muecas, esto se explica más detalladamente en 6.2.2.

Ensamblaje final

Tras realizar algún retoque en la idea inicial del diseño, como por ejemplo en el cuerpo, donde se ha eliminado la mochila como complemento, se ha apostado por una única pieza más estilizada. En general el resto del diseño se ha mantenido en el ensamblaje final.



(a) Vista Diagonal (b) Vista trasera (c) Vista frontal

Figura 2.6: Ensamblaje final

2.4.2. Diseño electrónico

El diseño electrónico del sistema [3], al igual que el diseño mecánico, se divide en dos partes principales, una para controlar los sistemas de navegación del robot (motores y sensores utilizados exclusivamente para la navegación) y otra para el control de las funciones de interacción social del robot.

Placa controladora GPMRC

Para el control de la base de Sacarino se ha desarrollado una placa controladora específica denominada GPMRC (General Purpose Mobile Robot Controller). Compuesta de dos microcontroladores dspic dsPIC30F6011A en configuración maestro-esclavo, es la encargada de realizar todos los controles de movimiento del robot (avance, retroceso y giro del robot), recibiendo las ordenes directamente del PC de alto nivel, y poniendo a disposición de éste toda la información sensorial del robot (lecturas de sonars, detección de marcas IR, odometría,...).

Se ha elegido este tipo de placa base para el control de todos estos procesos para poder realizar de forma rápida y eficiente las tareas de más bajo nivel que se precisa que sean atendidas en tiempo real. De esta forma se eliminan los efectos secundarios de utilizar un sistema operativo convencional proporcionando al sistema en general una cierta robustez y velocidad en su funcionamiento y disminuyendo la posibilidad de cuelgues debidos al software.

La arquitectura de control electrónico incluye dos placas controladoras de motores que se encargan de los lazos de control de velocidad y posición de los motores de tracción y dirección respectivamente. Aunque en el diseño original del robot era la placa GPMRC la encargada de controlar los motores directamente, en la primera fase de modificaciones se optó por esta configuración. Las placas controladoras de motores se conectan a través del BusCAN con la controladora GPMRC de la cual reciben la referencia de velocidad o posición que deben mantener según el motor que controlen. A su vez informan a la controladora central sobre el estado de los motores, en concreto con respecto a su velocidad y a la posición.

Esta arquitectura presenta dos ventajas fundamentales:

La comunicación de las placas de control de motores a través del BusCAN permite situarlas más cerca de los drivers de los motores reduciendo las probabilidades de ruido eléctrico en las señales de control y agilizando los ciclos de control. Dado que los lazos de control se ejecutan en los microcontroladores de estas placas, se liberan recursos computacionales en la controladora principal GPMRC

La placa de control GPMRC, envía la referencia de velocidad o posición que deben mantener cada una de las placas y éstas a su vez informan a la controladora central sobre el estado de los motores, de forma que la controladora central sigue encargándose de recoger toda la información de bajo nivel del robot y transmitirla hacia las capas más altas de control. Las funciones generales que realiza la GPMRC son [3]:

1. Supervisar la alimentación del sistema: Para este objetivo la GPMRC mide constantemente el nivel de la batería e informa si este se encuentra bajo, si el nivel es crítico realiza paradas de emergencia y activa una alarma para la carga sea inmediata.

2. Ejecutar paradas de emergencia: La placa ejecuta paradas de emergencia y activa diferentes alarmas al detectar la activación de las setas de emergencia, nivel de batería crítico o algún fallo electrónico.

3. Controlar el encendido del sistema informático: La placa se encarga de controlar la alimentación del PC y activa una alarma en caso de pérdida de comunicaciones con la placa.

4. Calcular la posición odométrica del sistema: A partir de la información de los encoder la placa calcula la posición odométrica de robot con respecto al punto de arranque del sistema, o a punto ordenado por el sistema informático.

5. Centralizar toda la información correspondiente al estado del robot y transmitirla al ordenador: La placa recibe la información proveniente de la concentradora de sensores la cual utiliza para actualizar el estado del robot junto con la información percibida por la misma GPMRC y retransmite toda la información a las capas de control superiores.

6. Controlar los datos presentados en el display LCD. Permite ver los datos principales del estado hardware y alarmas sin necesidad de que el PC esté encendido.

Placa de sensores

Debido al diseño mecánico, que permite el giro de la base giratoria sobre el sistema de tracción, se presentan una serie de restricciones mecánicas que limitan la cantidad de señales que pueden enviarse desde los sonars y bumpers, situados en los paneles laterales de la base giratoria, hasta la placa GPMRC, ubicada en el sistema de tracción. Por este motivo se ha desarrollado una placa de circuito impreso denominada concentradora de sensores para la lectura de los anillos de sonars y bumpers del robot. La placa toma medidas de los sonars y bumpers, según un patrón de captura predefinido, y se comunica con la GPMRC a través del bus CAN [3].

La placa concentradora está formada por un microcontrolador dsPIC30F6011A, que lee 16 bumpers, 16 sonars y controla relé de apertura y cierre del cajón frontal. Se comunica con el Maestro de la GPMRC por bus CAN. Mediante 16 bits el PC, a través del Maestro, puede seleccionar los sonars y los grupos de sonars que quiere que la concentradora lea, y este registro lo almacena en una memoria eeprom interna. Las funciones de la concentradora son:

- Leer las distancias medidas por el anillo de 16 sonars de la plataforma móvil del robot.
- Leer el estado de los bumpers de los paneles laterales del robot.
- Controlar la apertura y el cierre de la puerta del cajón dispensador.
- Comunicarse con la GPMRC a través de protocolo buscan.

Tarjeta driver de servomotores

La placa controladora de servos se encarga de los motores que mueven las articulaciones de hombro y codo de ambos brazos, los dos motores que inclinan y giran la cabeza respectivamente, y los párpados. El microcontrolador central es un pic 16F876

que genera las señales PWM respectivas a cada servo en función de lo que recibe por el puerto RS232/USB desde el PC. Esta placa está situada en el interior del torso del robot [3].

La controladora gobierna un total de 6 servomotores (5 de ellos los alberga la cabeza). Tres de ellos son dedicados a ganar 3 grados de libertad para el cuello, uno situado fuera de la cabeza y los dos restantes montados con geometría diferencial en la cavidad interna, mientras que de los tres restantes, dos se dedican al movimiento independiente de párpados y otro para ambas cejas.

2.4.3. Subsistemas

El sistema operativo de Sacarino II es ROS Indigo, lo que es importante, puesto que favorece el diseño modular por subsistemas, esto es, que se puede programar cada funcionalidad de manera independiente, a continuación se describen algunos de los subsistemas de Sacarino II:

Subsistema de seguridad

Se encarga de detectar posibles obstáculos que se presenten a lo largo de la trayectoria y así evitar colisiones a partir de los datos obtenidos de los ultrasonidos y el láser. En el peor caso recibiría la señal de activación de los bumpers situados en el panel lateral que rodea la base, indicando que se ha producido contacto. Una vez va comprobando las señales de los sensores ya mencionados, si hay que hacer alguna modificación de la ruta, comunicaría las alarmas a los módulos de navegación para que tomen las acciones pertinentes o directamente al Control Plataforma para que detenga el movimiento.

Comunicación verbal

Se define como la capacidad que tiene el robot de poder comunicarse con los usuarios de manera verbal. Incluye desde la comunicación de información a través de voz (Síntesis de sonido), hasta reconocimiento de voz y la posibilidad de manterner una conversación

más o menos fluida (Gestión de Diálogo). Esta capacidad dota a Sacarino II de un rasgo fundamental en el parecido a un humano.

Figura 2.7: Reconocimiento de voz en la interfaz

Interfaz

En la parte frontal del cuerpo de Sacarino II se encuentra una pantalla táctil donde se muestra un interfaz simple (se ampliará en concepto de interfaz en el capítulo 7) que permite la interacción con agentes externos. La interfaz ha sido desarrollada utilizando las librerías QT de código abierto e integrada en un módulo ROS. Todas las interacciones se realizan de manera bidireccional mediante la interfaz multimodal que incluye voz y pantalla táctil. Sacarino II presenta la información mediante voz y texto escrito en la pantalla, imágenes, etc. Por otra parte, el usuario puede solicitar información también a través de la pantalla táctil directamente pulsando en la opción deseada o activando el Reconocimiento de Voz en el botón correspondiente [3].

Figura 2.8: Interfaz [3]

Navegación

Comprende a su vez una agrupación de subsistemas [3]:

- Localización por Odometría:Calcula la posición estimada en base al movimiento del robot con los datos recibidos desde el módulo Control Plataforma (encoders) y del nodo Wii Motion + (giróscopo).
- Nodo Láser: Driver encargado de gestionar el láser sick LMS100 y obtener los datos de escaneo a través de la red TCP/IP y transformarlos a mensaje ROS.
- Localización Mapas: Estima la posición del robot mediante distribución probabilística utilizando el algoritmo de localización adaptativo de Monte-Carlo (AMCL) [15], a partir de los datos de odometría (Localización por Odometría) y los escaneos láser (Nodo Láser), que son comparados con un mapa del entorno predefinido. Los fundamentos del funcionamiento de este sistema se explican más detalladamente en el apartado de Nivel de aplicación.
- Navegación Mapas: Se encarga de llevar a cabo el seguimiento de trayectorias cuando el sistema de localización sea el de mapeado láser realimentado por la posición recibida del módulo Localización Mapas. Escucha órdenes de inicio, pausa y reanudación del seguimiento de la trayectoria que puede recibir a modo de serie de puntos o leerla directamente desde un archivo. Al recibir desde el Sistema de Seguridad el aviso de presencia de obstáculos detiene su marcha, continuándola únicamente cuando el obstáculo ha dejado el camino libre. Aunque los sistemas de Planificación y Navegación por Mapas podrían permitir la evitación de obstáculos recalculando la trayectoria, la experiencia ha determinado que es más eficiente este tipo navegación reactiva más conservadora.
- Gestor de Navegación: Es el encargado de recibir las tareas que requieran movilidad de la plataforma desde el módulo Secuenciador. Una vez establecido el punto de destino y conocida la posición actual, planifica a partir de un mapa topológico, las sub-tareas de navegación requeridas (maniobras de conexión/desconexión, seguimiento de trayectoria,). Después comunica al módulo correspondiente el inicio y fin de cada sub-tarea y supervisa su cumplimiento.

Control Gestual

Sacarino II tiene capacidad para simular diferentes emociones o gestos que le confieren expresividad y una mejor interacción con el usuario. Todos estos gestos vienen marcados por el estado en que se encuentra el robot en cada momento por lo que dependen principalmente del módulo de Comportamientos, y del Secuenciador que marca las diferentes acciones a realizar, después el Control Gestual comunica al Control Cuerpo las diferentes sub-acciones en las que se descompone un gesto completo. De esta forma el robot es capaz de dormirse si no hay nadie cerca, mover la boca cuando habla, o mover el cuello para mirar fijamente al usuario con quien está interactuando.

Capítulo 3

SISTEMAS DE VISIÓN ARTIFICIAL

El presente capítulo está dedicado a la tecnología de visión artificial, evolución y actualidad, estudio en profundidad de las diferentes tecnologías actuales para visión 3D y elección del sensor de visión artificial más adecuado para la realización del proyecto.

3.1. Evolución de la visión por computador

La visión artificial comienza en los años 60, con la simple idea de conectar una videocámara a un computador; esto implica que lo que interesa no es solamente la captura de la imagen, sino comprender lo que la imagen representa. Uno de los resultados más destacables fue un programa desarrollado por Larry Roberts (creador de ARPAinet) «mundo de microbloques» con el que el robot podía observar una estructura de bloques sobre una mesa, analizar su contenido y rehacerla desde otra perspectiva, demostrando que el computador era capaz de procesar la imagen.

A partir de entonces, los primeros sistemas se basaban en imágenes binarias que se procesaban en bloques, píxeles o ventanas, pero con el imparable crecimiento del poder de procesamiento de los microprocesadores, se han desarrollado nuevos algoritmos y técnicas dentro del campo de la visión artificial. Y, gracias a esto se logró reconocer contornos de objetos y su posición dentro de una imagen, pero estos algoritmos tenían el inconveniente de no poder trabajar con diferentes tipos de iluminación.

Posteriormente, se introdujeron los sistemas de intensidad de grises en estos, con los cuáles cada píxel era representado de acuerdo a un número proporcional a la intensidad de gris de dicho elemento. Con esta técnica se podría operar en lugares con distinta iluminación, puesto que se podían encontrar los bordes de los objetos utilizando cambios en los valores de intensidad de los píxeles.

Actualmente, la visión por computador comprende tanto la obtención como la caracterización e interpretación de los objetos contenidos en una imagen y es uno de los elementos sensoriales más atractivos para asegurar la autonomía en los robots ya que proveen de información relevante sobre el estado de éstos y de su entorno físico más inmediato, además de ser más baratos en comparación con otros; el único reto que plantean es cómo extraer información de una imagen.

3.2. Visión 3D

En el área de la robótica de servicios detallada en el capítulo anterior, se hace necesario no sólo procesar los datos de una sola imagen, sino que resulta mucho más interesante y necesaria la obtención de información del entorno en tres dimensiones con el que el robot se va enfrentando, como si de un humano se tratase.

La obtención 3D de una escena se puede realizar mediante técnicas muy diversas. Estas técnicas se dividen según [Rocchini et al.,2001], en dos grandes grupos: adquisición con contacto y sin contacto. Las técnicas de adquisición con contacto se dividen a su vez en técnicas destructivas, como el laminado, y en técnicas no destructivas, como los brazos articulados. Por su parte, las técnicas sin contacto se subdividen en técnicas transmisivas y reflectivas. Además, las técnicas reflectivas agrupan, por un lado, a las técnicas no ópticas,como radar y sonar, y, por otro, a las técnicas ópticas. Por tanto, las técnicas ópticas de visión artificial son un subconjunto dentro de las técnicas que se pueden utilizar para obtener información tridimensional en un escena.

Las técnicas sin contacto se exponen en la figura 3.1. Concretamente, ampliaremos a lo largo del capítulo los principios de los métodos de tiempo de vuelo, métodos estéreo y técnicas de luz estructurada.

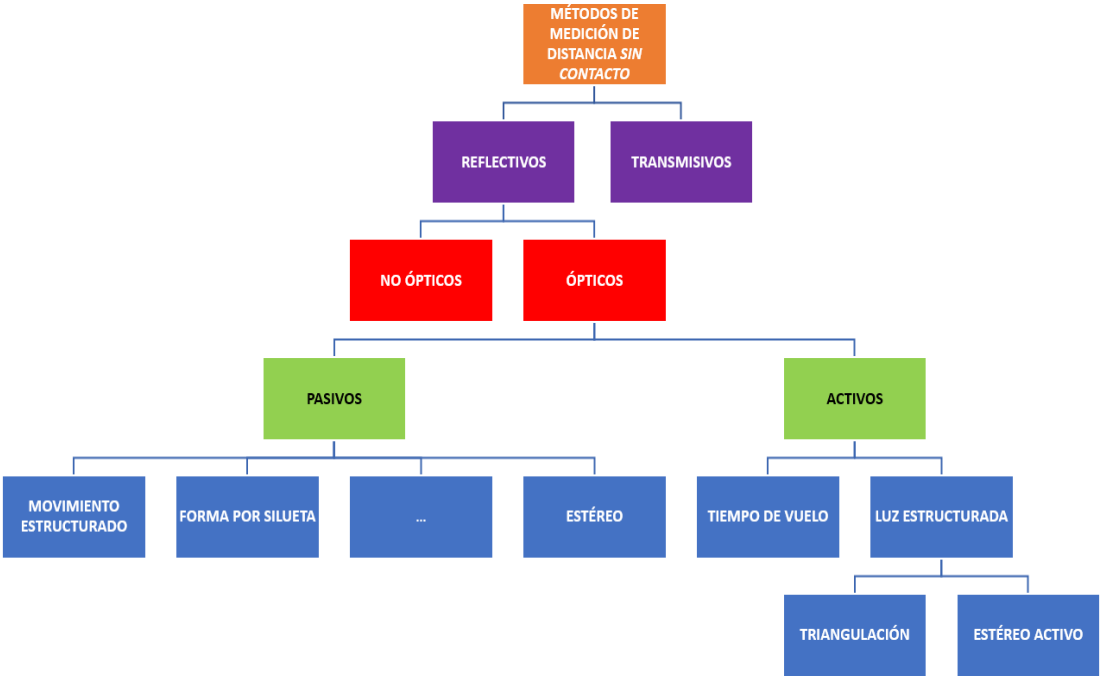


Figura 3.1: Taxonomía de los métodos 3D (adaptado de [4])

3.2.1. Modelo de cámara pin-hole. Matrices de Proyección. Triangulación

Para entender el funcionamiento de las tecnologías 3D conviene definir previamente conceptos como modelo de cámara pin-hole, matrices de proyección y triangulación.

Como se expone en [4], consideramos un sistema de referencia 3D (con ejes x,y,z) llamado Sistema de Coordenadas Cámara (CCS), con origen en O, llamado centro de proyección y un plano paralelo al x-y intersecando al eje z negativo en la coordenada f al que nos vamos a referir como plano sensor. Además, considerar el siguiente sistema de referencia 2D:

$$u = x + c_x \tag{3.1}$$

$$v = y + c_y \tag{3.2}$$

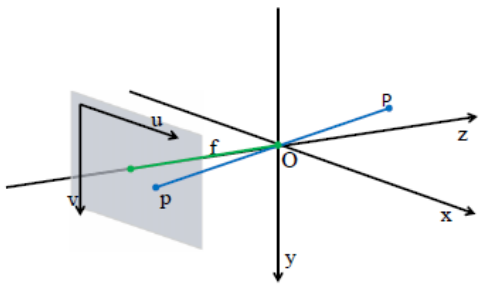


Figura 3.2: Perspectiva de la proyección (de [4])

asociado al plano sensor, orientado como en la figura 3.2. La intersección c del eje z con el plano sensor da como coordenadas c =[u=c$_x$, $v = c_y$]^T. El conjunto de puntos p, llamado píxeles, de coordenadas $\mathbf{p}$ = [u,v]^T obtenido de la intersección de los rayos que conectan el centro de proyección O con todos los puntos del espacio 3D de la forma $\mathbf{P}$ = [x,y,z]^T es la «impresión» de la escena en el sensor S. La relación entre P y p llamada proyección central, la podemos obtener por similitud de triángulos.

$$\begin{cases} \text{u - } c_x \equiv fx/z \\ \text{v - } c_y \equiv fy/z \end{cases} \tag{3.3}$$

Donde f es la distancia focal, esto es entre el centro de proyección O y el plano sensor. En la bibliografía esta distancia está denotada como -f, por cortar en una cota negativa al eje z, pero nosotros mantendremos la notación de f positiva. Las ecuaciones 3.3 son una buena descripción de la relación entre las coordenadas de la escena y aquellas que se obtienen como imagen de ella mediante un dispositivo pin-hole (con una agujero practicado) situado en el centro de Proyección O. Este sistema permite que un solo haz de luz vaya a través del pin-hole a O. Por varias razones es habitual el uso de ópticas, p ej, lentes, en vez de pin-holes. Si una lentilla delgada sustituyera al pin-hole, los ejes ópticos coincidirían con el eje z del CCS.

Parámetros intrínsecos y extrínsecos de una cámara

La proyección geométrica asocia a cada punto p del plano sensor 2D de coordenadas cartesianas $\mathbf{p}$ = [u,v]^T un punto 3D, que en coordenadas homogéneas toma la forma $\tilde{p}$ = [hu,hv,h]^T donde h es una constante real. Tomar h = 1 es una práctica habitual y [u,v,1]^T se conoce habitualmente como *vector extendido de p*.Por tanto, podemos interpretar que al vector como el rayo que une el centro de proyección O con

el correspondiente píxel p del plano sensor. De forma similar, cada punto del espacio tridimensional que adopta una forma del tipo $\mathbf{P} = [\text{x,y,z}]^T$ admite una representación en la forma homogénea $\tilde{P} = [\text{x,y,z,1}]^T$. La representación homogénea permite entonces reescribir las expresiones (3.3) en la correspondiente forma matricial, quedando:

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.4}$$

Nótese que la parte izquierda de la expresión (3.4) denota la representación en 2D de p y la parte derecha de la igualdad el punto P en coordenadas cartesianas 3D.

Los sensores digitales son típicamente matrices planas de células sensibles rectangulares que portan sistemas fotoeléctricos de conversión como las tecnologías CMOS o CCD en el caso de video-cámaras o foto-cámaras o un receptor de tiempo de vuelo en el caso de cámaras ToF. Dada una dimensión finita de un plano sensor de N_R x N_C normalizadas con origen en (0,0) y coordenadas píxel unitarias $u_s \in [0, \ldots, \text{N}_C \text{ - } 1]$ y $v_s \in [0, \ldots, \text{N}_R \text{ - } 1]$ en ambas direcciones u y v, podemos reescribir la expresión (3.4) como:

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.5}$$

Donde K es la matriz de parámetros intrínsecos definida como:

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Con $\text{fx} = \text{f}k_u$ la distancia focal sobre el eje x de la óptica,$\text{fy} = \text{f}k_v$ la distancia focal sobre el eje y, c_x y c_y las coordenadas (u,v) de la intersección del eje óptico con el plano sensor. Todas estas cantidades están expresadas en [píxel], esto implica que si f está en mm, k_u y k_v se expresan en [píxel]/[mm]. En ocasiones interesa expresar las

coordenadas del punto P no respecto al Sistema de coordenadas Cámara sino, a otro sistema de referencia, conocido como sistema de coordenadas mundo, procediendo de manera análoga se puede demostrar que si la transformación entre sistemas es la suma de una rotación R mas una traslación T, entonces:

$$z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = KP = K[RT]\tilde{P_W} = M\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.7}$$

Como consecuencia de distorsiones y aberraciones de las ópticas reales, las coordenadas $\mathbf{p} = (\hat{u}, \hat{v})$ del píxel asociadas al punto de la escena **P** de coordenadas $\mathbf{P} = [\text{x,y,z}]^T$ en el sistema CCS no satisfacen la relación 3.5. Las coordenadas que realmente están asociadas a **P** se pueden hallar aplicando modelos de distorsión como por ejemplo $p_T = \Psi^-1(\hat{p_T})$, donde Ψ denota la transformación de la distorsión. Modelos anti-distorsión como el conocido *Heikkila model* se adaptan muy bien a la mayoría de sistemas de imagen y se muestran efectivos en la computación de los parámetros:

$$\begin{bmatrix} u \\ v \end{bmatrix} =$$

$$\Psi^-1(\hat{p_T}) = \begin{bmatrix} \hat{u}(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2d_1\hat{v} + d_2(r^2 + 2\hat{u^2}) \\ \hat{v}(1 + k_1r^2 + k_2r^4 + k_3r^6) + d_1(r^2 + 2\hat{v^2}) + 2d_2\hat{u}) \end{bmatrix} \tag{3.8}$$

donde $r = \sqrt{(\hat{u} - c_x)^2 + (\hat{v} - c_y)^2}$, los parámetros k_i con i = 1,2,3 son constantes de la distorsión radial y d_i con i = 1,2 constantes de la distorsión tangencial. Los parámetros de distorsión $\mathbf{d} = [k_1,k_2,k_3,d_1,d_2]$ son parámetros intrínsecos que tienen que ser considerados conjuntamente con $[\text{f},k_u,k_v,c_x,c_y]$.

3.2.2. Cámaras por tiempo de vuelo. Principio y funcionamiento

La base de los sensores por tiempo de vuelo (o RADAR, Radio Detection And Ranging) es la estimación de la distancia radial a un punto de una escena mediante el

tiempo que tarda la radiación electromagnética en viajar desde el sensor a la escena y volver, a la velocidad de la luz, aproximadamente c $\cong 3 \times 10^8[m/s]$. Así, la distancia cubierta ρ cubierta en un tiempo τ por la radiación óptica es $\rho = c\tau$.

La figura 3.3 muestra el esquema típico de medición por tiempo de vuelo, la radiación emitida en tiempo 0 por el transmisor del sensor ToF (Time of Flight) situado a la izquierda viaja directo a la escena cubriendo una distancia ρ, que a continuación es reflejada de vuelta por la superficie de la escena y que de nuevo viaja idealmente otra distancia ρ de vuelta al sensor, alineado con la escena. Por lo que, la relación entre ρ y τ es (3.9), que resulta el principio básico de funcionamiento.

$$\rho = \frac{c\tau}{2} \tag{3.9}$$

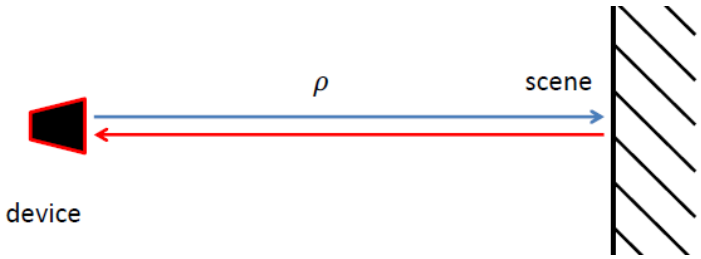


Figura 3.3: Principio de Sensores Time of Flight

Sin embargo, lo que habitualmente interesa medir son las superficies de la escena más que puntos discretos y aislados. Para ello, normalmente se monta a los sensores ToF en plataformas con el objeto de ir escaneando la escena y obtener como salida una nube de puntos, en concreto si el movimiento que describen es lineal, se conocen como sistemas LIDARs (Light Detection And Ranging).

A pesar de que estos sistemas ofrecen mayor robustez, hay cierto tipo de escenas que son intrínsecamente incompatibles con la medición por tiempo de vuelo, por ejemplo aquellas en las que los objetos están moviéndose (escenas dinámicas). A diferencia de los sistemas que adquieren la geometría de la escena por el escaneo secuencial de puntos para conformar una nube, existen también las cámaras matriciales por tiempo de vuelo, que estiman la geometría de la escena en un solo disparo por medio de una matriz de

$N_R \times N_C$ (Nr número de filas y Nc número de columnas) en la que cada elemento (píxel) mide la distancia a la escena, fenómeno que se produce simultáneamente para todos los píxeles en un solo disparo.

Las cámaras por tiempo de vuelo ofrecen como salida mapas de profundidad a frecuencias de video, o matrices de medición de distancias si se las da como entrada la distancia entre el píxel de la matriz y el punto de la escena.

Aunque a priori, la ecuación (3.9) es simple, su implementación no es nada sencilla y esconde tremendos retos porque involucra la velocidad de la luz. Por ejemplo, de acuerdo a (3.9), se tardaría 5[ps] en cubrir[1mm] de distancia, por lo que los sistemas de medición cuya resolución en distancia es 1mm nominal, necesitarían un reloj capaz de medir el pulsos de tiempo de 5[ps]. La implementación de este tipo de dispositivos y su integración en la configuración matricial del sensor son de suma importancia en el desarrollo de sistemas por tiempo de vuelo.[1]

Diferentes tecnologías en la implementación del reloj lleva a diferentes tipos de cámaras ToF. Las elecciones mas comunes hoy en día son de «ola continua»(CW) , por modulación de intensidad, por shutter óptico(OS), o por SPAD (single-photon avalanche diodes).

3.2.3. Sistemas de visión estéreo. Principio y funcionamiento

Un sistema de visión estéreo (del inglés stereo), en general está compuesto por dos cámaras, típicamente idénticas que comparten parcialmente una escena, además de ser coplanares, tener alineados los planos sensores y ejes ópticos alineados. La teoría tras la calibración y rectificación, llamada «geometría epipolar» de estos sistemas puede ser encontrada en libros clásicos de visión artificial como [16] o [17].

Vamos a introducir una notación básica para los sistemas estéreo. Las dos cámaras que conforman el sistema de visión stereo S son, la cámara izquierda L(también conocida como *cámara referencia*) y la cámara derecha R (conocida como *cámara objetivo*). Cada cámara, tiene su propio sistema de referencia 3D y en su plano sensor situado un sistema

[1] Como se desarrolla en [4]

de referencia 2D, tal y como se muestra en la figura 3.4.

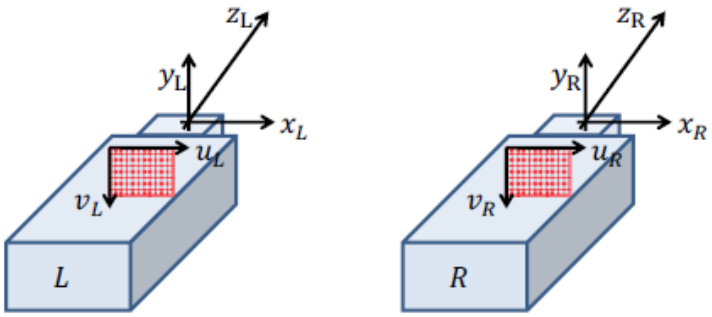


Figura 3.4: Sistemas de referencia en sistemas stereo (de [4])

Nombraremos al sistema de referencia 3D de la cámara izquierda con la terna (x_L, y_L, z_L) y al sistema de referencia 2D del plano sensor izquierdo como (u_L, v_L).A su vez, el sistema de referencia de la cámara derecha como (x_R, y_R, z_R) y el sistema de referencia del plano 2D derecho como (u_R, v_R). Una convención generalmente aceptada es considerar el sistema de referencia 3D de la cámara izquierda L-3D como el sistema de referencia de todo el sistema estéreo, y denotarlo por tanto como S-3D.

Si partimos de un sistema estéreo calibrado y rectificado entonces, un punto en el espacio 3D de coordenadas $\mathbf{P} = [\mathrm{x,y,z}]^T$ con respecto al sistema de referencia S-3D, está proyectado en los píxeles p_L y p_R de las cámaras L y R con las coordenadas $p_L = [\mathrm{u}_L, v_L]^T$ y $p_L = [\mathrm{u}_R = u_L - d, v_R = v_L]^T$ respectivamente, como se ilustra en la figura . Fruto de las relaciones que se derivan al formar un triángulo con vértices en p_R, P y p_L, se puede demostrar que, tras rectificación, donde los puntos p_L y p_R comparten la misma componente vertical, la diferencia entre las componentes horizontales d = u_L - u_R, (llamada *disparidad*), es inversamente proporcional al valor de la profundidad z de P a través de la sabida relación de triangulación

$$z = \frac{b|f|}{d} \tag{3.10}$$

Al lector interesado en saber la deducción de la fórmula (3.10), le referimos a [16], [17]. En (3.10) b es la línea base, es decir, la distancia entre los orígenes de los sistemas de referencia L-3D y R-3D y | f | es la distancia focal de las cámaras. Los píxeles p_L y p_R son conocidos como «conjugados». De las coordenadas 2D de p_L y su profundidad

asociada z obtenida de (3.10), las coordenadas x e y del correspondiente punto 3D pueden ser fácilmente calculadas si invertimos la ecuación de proyección (figura 3.5) relativa a la cámara L, esto es:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = K_L^{-1} \begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix} z \tag{3.11}$$

Donde K_L^{-1} es la inversa de la matriz de parámetros intrínsecos rectificada de la cámara L. Por tanto, resumiendo, una vez que obtenemos los píxeles conjugados p_L y p_R de una pareja de imágenes estéreo, es inmediato reconstruir los puntos 3D de coordenadas $\mathbf{P} = [\text{x,y,z}]^T$ de la escena a través de (3.10) y de (3.11), este proceso es conocido como *triangulación* o *stereoscopía computacional*.

La disponibilidad de un par de píxeles conjugados es la parte que presenta una mayor dificultad del proceso descrito, e incluso para empezar puede no existir debido a oclusiones y si existe, puede que no sea tan directo el encontrarlos. Detectar píxeles conjugados (u homólogos) en un par de imágenes estéreo, a menudo llamado «proceso de correspondencia» , es uno de los mayores retos de un algoritmo de visión estéreo. Los métodos propuestos para abordar este problema suelen estar divididos en aproximaciones locales y globales. Los métodos locales consideran únicamente similitudes en mediciones locales entre el entorno de p_L y las regiones que presentan parecido de todas las candidatas en toda la misma fila de p_R. El conjugado que sea seleccionado es aquel que maximiza la similitud, un método típicamente conocido como «Winner Takes All» (estrategia WTA).

Los métodos globales no consideran cada par de puntos en sí mismos, sino que estiman todos los valores de disparidad en una vez aprovechándose de esquemas de optimización global. Estos métodos están basados en formulaciones Bayesianas a las que se está prestando gran atención en la actualidad. Estas técnicas generalmente modelan la escena como un campo aleatorio de Markov (MRF) en el que se incluye un único marco de trabajo fruto de combinar comparaciones locales de las dos imágenes con restricciones de cambios suaves en la profundidad. Los algoritmos de métodos es-

téreo globales estiman la disparidad de la imagen minimizando una función de costes compuesta por dos términos, un término de datos que representa el coste de correspondencias locales, de manera parecida a un algoritmo local (p.ej: covarianza) y de un término de suavidad (esto es evitar cambios bruscos en la profundidad) definiendo el nivel de suavidad de la disparidad a partir de ir contando discontinuidades, explícita o implícitamente.

Aunque algoritmos específicos pueden haber generado un impacto importante en la solución al problema de la correspondencia, la calidad final de la reconstrucción 3D de un sistema estéreo sigue dependiendo inevitablemente de las características de la escena. Para ver esto, se puede considerar el caso de una escena sin rasgos geométricos ni color, como una pared de color uniforme. Las imágenes estéreo de dicha escena serán uniformes y, por tanto, no se podrán obtener píxeles conjugados de ella. Además, no se podrá obtener ninguna información sobre la profundidad de la triangulación.

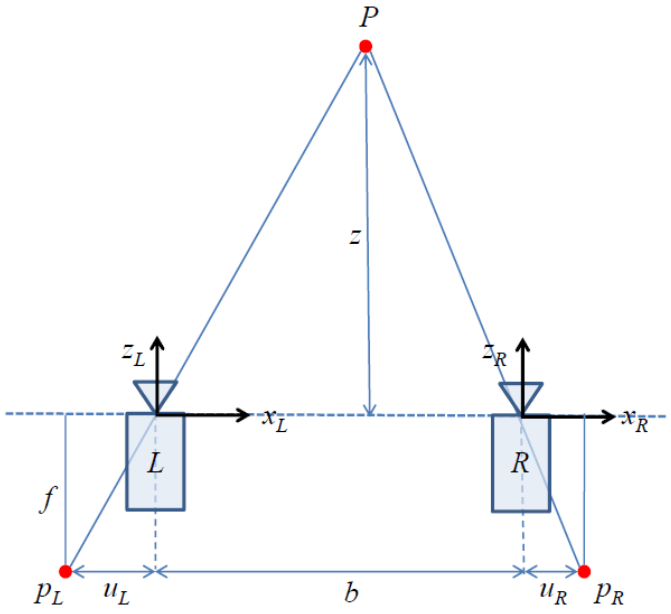


Figura 3.5: Triangulación con un par de cámaras alineadas y rectificadas (de [4])

La triangulación activa, también llamada «técnica de luz estructurada», ofrecen una manera efectiva de enfrentar estos problemas de correspondencia.

3.2.4. Luz estructurada. Principio y funcionamiento

La ecuación (3.10) deriva de la relaciones triangulares mostradas en la figura 3.5. El hecho de que existan p_L y p_R en sistemas de visión estéreo es por la luz reflejada en un punto P del entorno 3D, pero es secundario. Lo que verdaderamente importa es la geometría triangular entre los rayos Pp_L,Pp_R y $pLpR$. Desde esta perspectiva en la que puntos de la imagen son equivalentes a rayos salientes de dos centros de proyección, cualquier dispositivo capaz de proyectar rayos entre su centro de proyección y los puntos de la escena puede ser equivalente funcionalmente a una cámara estándar. Este es el caso de los proyectores de luz, que para cualquier rayo que emiten, conecta su centro de proyección a el punto de la escena P por el haz de luz que comienza en Pp_A y que es proyectado hacia P, como se muestra en la figura .

Un sistema de visión estéreo donde una de sus dos cámaras es sustituida por un proyector, p.ej: hecho por una supuesta cámara C y un proyector A mostrado en la figura 3.6, se conoce como *activa* o *luz estructurada*. La cámara C tiene un sistema de referencia (x_C, y_C, z_C), también llamado sistema de referencia C-3D y un sistema de referencia 2D con coordenadas (x_C, y_C), como el fig 3.6. De manera análoga, el proyector A tiene su sistema de referencia 3D con coordenadas (x_A, y_A, z_A), conocido como sistema de referencia A-3D y un sistema 2D de coordenadas (u_A, v_A). Al igual que los sistemas estéreo pasivos, los sistemas activos también pueden ser calibrados y rectificados [18] de cara a simplificar el proceso de estimación de profundidades.

La figura 3.6 muestra la proyección de un haz de luz que parte de p_A con coordenadas $p_A = [u_A, v_A]^T$ en el sistema de referencia A-2D, sobre un punto P 3D de la escena, con coordenadas $\mathbf{P} = [\mathrm{x,y,z}]^T$ en el sistema de referencia C-3D. Si P es un punto no ocluido, entonces la componente especular del haz será reflejada en P produciendo otro rayo que alcanzará el píxel p_C en la cámara C, formándose el triángulo p_CPp_A. Si el sistema activo está rectificado y calibrado, p_C tendrá coordenadas $p_C = [u_C = u_A + d, v_C = v_A]^T$. Como en el caso de los sistemas estándar estéreo, p_A y p_C serían píxeles conjugados, por lo que una vez sus coordenadas son conocidas, podemos hallar el valor de la profundidad z de P mediante (3.10), que en este caso es conocida como *triangulación activa* por ser A un sistema activo. Las coordenadas 3D de P pueden ser obtenidas de aplicar (3.11)

como en el apartado anterior.

La efectividad de los sistemas activos respecto al problema de correspondencia es fácilmente apreciable si consideramos el ejemplo del apartado anterior en el que se presentaba una pared lisa de color uniforme. En este caso, A proyecta un rayo sobre un punto P de la escena, iluminándolo y resultando su color, que es inmediatamente devuelto a C. Por lo tanto, es fácil deducir que P queda reconocido al tener un color distinto al resto de su vecindad. En este caso p_C y p_A son reconocibles y existen.

Los sistemas de luz estructurada pueden, además, proporcionar información sobre la profundidad de escenas sin rasgos geométricos ni de color, a diferencia de los sistemas estéreo pasivos, que fallaban al intentar dar una profundidad en estos casos.

Las características del patrón de luz proyectado son fundamentales a la hora de poder resolver el problema de correspondencia y ofrecer un buen rendimiento, por eso es un campo de investigación y desarrollo en la actualidad. El uso de secuencias de varios patrones de luz ha sido típicamente usado en los principios de estas técnicas y esto provocaba que el umbral al que se podían medir profundidades fuese muy limitado. En general, las tecnologías activas son más caras y algo más lentas que los métodos pasivos, pero bastante más exactos y robustos que estos. Si nuestro objetivo es medir profundidades en escenas dinámicas, con objetos en movimiento, personas etc, entonces necesitaremos estos sistemas activos, con la peculiaridad de que existen métodos que perfeccionan la respuesta, por ejemplo reducir el número de patrones proyectados para aumentar la rapidez. En el siguiente apartado se profundizará en el estudio del sensor kinect.

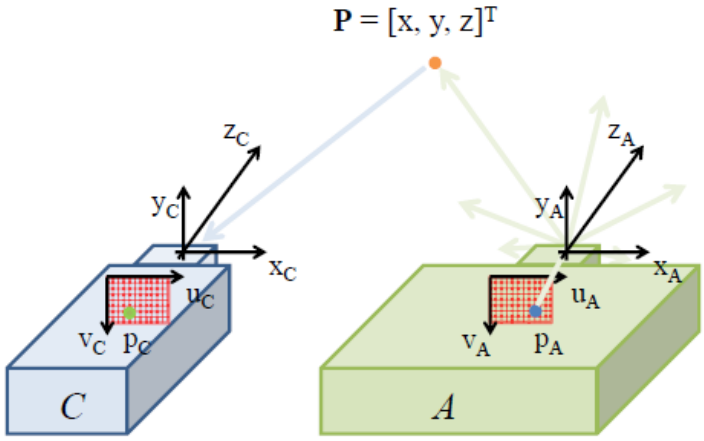


Figura 3.6: Triangulación activa

3.2.5. Kinect como sistema de Visión Artificial

Kinect ha llegado a ser un importante sensor 3D. Ha sido el foco de atención de numerosos desarrollos debido al rápido reconocimiento de posturas humanas, su bajo coste, su alta fiabilidad y alta velocidad de medición, que seguro sitúan a Kinect como uno de los sensores 3D de referencia. Este apartado provee al usuario de un análisis del potencial de Kinect de acuerdo a [5].

Kinect es un dispositivo compuesto por un proyector infrarrojo, una cámara (IR) y una cámara color RGB. La cámara IR y el proyector son usados para lograr la triangulación activa con los puntos 3D de la escena. La cámara RGB, en general se usa para extraer texturas de la escena o para reconocimiento de contenido. Como equipo de medición, Kinect ofrece como salidas: una imagen IR, una imagen RGB y un mapa (inverso) de profundidad.

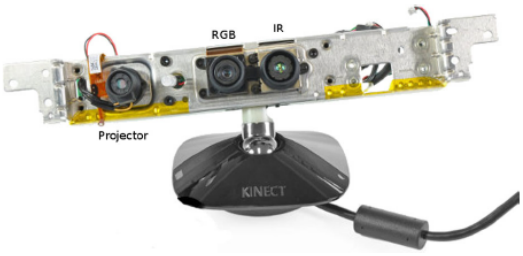


Figura 3.7: Sensor 3D Kinect(de [5])

Imagen IR

La cámara IR de la figura 3.8, tiene una resolución de 1280 x 1024 píxeles por un campo de visión de 57 x 45 grados, 6,1 mm de distancia focal, 5.2 micrómetros de tamaño de píxel. El uso de esta cámara se centra en observar y decodificar la proyección del proyector IR para triangular la escena 3D. Si la escena está suficientemente iluminada y se bloquea el proyector IR, la cámara RGB se puede calibrar de manera fiable usando un patrón de tablero de ajedrez. Además se tiene que añadir, que la cámara presenta distorsión radial y tangencial.

Imagen RGB

La cámara color RGB tiene una resolución de 1280 x 1024 píxeles para un campo de visión de 63 x 50 grados, 2,9 mm de distancia focal y 2.8 micrómetros de tamaño de píxel. Ofrece imágenes de calidad media. Puede ser calibrado mediante un sistema de tracking relativo a través de posiciones consecutivas de imágenes (sistema SfM).

Resolución del mapa de profundidad

La figura 3.8 muestra la resolución de la profundidad medida en función de la profundidad real. La resolución de la profundidad ha sido medida moviendo la Kinect de 0.5 m a 15 m a lo largo de un plano suficiente para grabar pasos con precisión. Todos los valores dentro de un campo de visión de aproximadamente 5º alrededor del centro de la imagen, como se expone en [5].

La cuantización del paso q [mm], que es la distancia entre dos valores grabados consecutivos, se ajusta a una ecuación de tipo cuadrática en función de la profundidad z[m].

$$q(z) = 2{,}73z^2 + 0{,}74z - 0{,}58 \tag{3.12}$$

Esta ecuación concuerda con lo que cabía esperar, puesto que el prototipo de la función resolución de la profundidad respecto al paso para dispositivos basados en trian-

gulación activa es de tipo cuadrático. Los valores del extremo del rango q(0.5m)=0.65 mm y q(15.7 m) = 685 mm están de acuerdo con lo que se expone en [19].

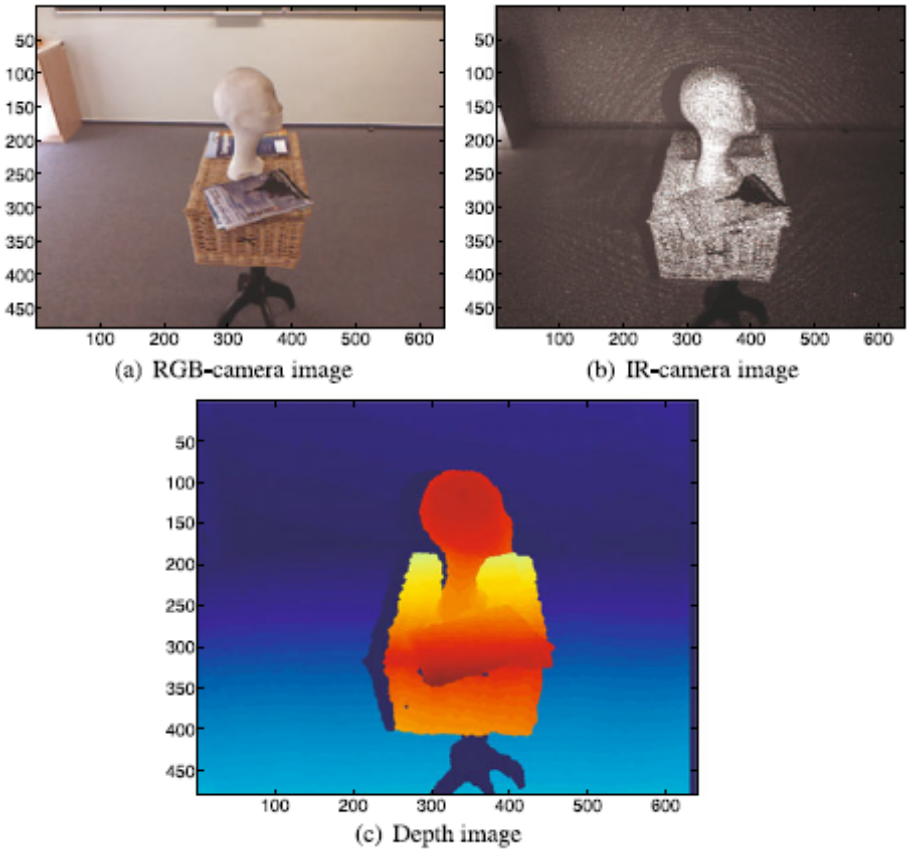


Figura 3.8: Imágenes de kinect (de [5])

Modelo geométrico de Kinect

Modelamos Kinect como un sistema multi-vista que consiste en cámaras RGB, IR y de profundidad. El modelo geométrico de las cámaras RGB e IR, que transforman un punto 3D X en un punto de una imagen $[u,v]^T$, y cuyo significado ha sido introducido en el apartado 3.2.1, viene dado por:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \tag{3.13}$$

$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = \underbrace{(1 + k_1 r^2 + k_2 r^4 + k_5 r^6) \begin{bmatrix} p \\ q \\ 0 \end{bmatrix}}_{\text{distorsión radial}} + \underbrace{\begin{bmatrix} 2k_3 pq + k_4(r^2 + 2p^2) \\ 2k_4 pq + k_3(r^2 + 2p^2) \end{bmatrix}}_{\text{distorsión tangencial}} \tag{3.14}$$

$$r^2 = p^2 + q^2, \quad \begin{bmatrix} pz \\ qz \\ z \end{bmatrix} = R(X - C) \tag{3.15}$$

Con parámetros de distorsión k =[k_1,k_2,...,k_5], matriz de calibración K, matriz de rotación R y matriz al centro de la cámara C. La cámara de profundidad en kinect, está asociada a la geometría de la cámara IR. Devuelve la inversa de la profundidad d a lo largo del eje z, como se puede ver en la figura 3.9, para cada píxel [u,v]^T de la cámara IR :

$$\begin{bmatrix} x \\ y \\ d \end{bmatrix} = \begin{bmatrix} u - u_0 \\ v - v_0 \\ \frac{1}{zc_1} - \frac{c_0}{c_1} \end{bmatrix} z \tag{3.16}$$

Donde u y v son obtenidas de (3.14), la verdadera profundidad z obtenida de (3.15), y u_0 y v_0 son factores de corrección de posicionamiento de los píxeles entre la cámara IR y la de profundidad, tras otros estudios en este mismo campo, una aproximación buena sería $u_0 = 4.1$ y $v_0 = 3.1$ [5]. X son las coordenadas 3D de un punto, y c_0 y c_1 parámetros del modelo. En general, asociamos el sistema de coordenadas de Kinect al de la cámara IR y, por tanto, obtenemos una matriz de rotación $R_{IR} = \mathrm{I}$ y una matriz $C_{IR} = 0$. Reconstruimos un punto 3D X_{IR} de la medida [x,y,d] en la imagen de profundidad mediante:

$$X_{IR} = \frac{1}{c_1 d + c_0} dis^{-1} \left(K_{IR}^{-1} \begin{bmatrix} x + u_o \\ y + v_0 \\ 1 \end{bmatrix}, \; k_{IR} \right) \tag{3.17}$$

y la proyección sobre las imágenes RGB como:

$$u_{RGB} = K_{RGB} dis(R_{RGB}(X_{IR} - C_{RGB}), k_{RGB}) \tag{3.18}$$

Donde «dis» es la función de distorsión dada por (3.14), k_{IR} es la matriz de calibración de la cámara y K_{RGB},R_{RGB}, C_{RGB} son las matrices de calibración, rotación y traslación al centro de la cámara RGB, respectivamente.

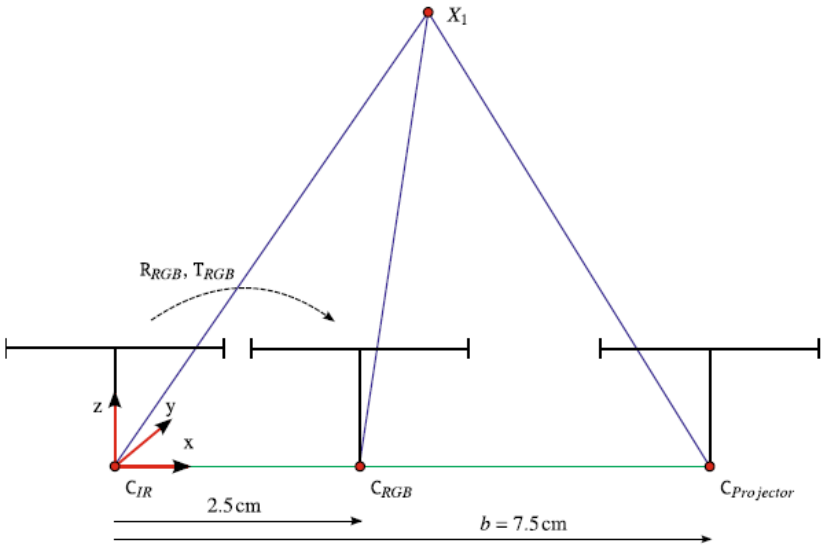


Figura 3.9: Modelo geométrico de Kinect (de [5])

Capítulo 4

PROGRAMACIÓN MODULAR, ROS

Hasta ahora se ha hablado de los últimos e innovadores avances dentro del campo de la sensorización, a nivel hardware. Un robot móvil social tiene que atender en tiempo real numerosos estímulos de su entorno constantemente, por lo que es de vital importancia apoyar toda la base hardware sobre un sistema operativo que gestione de manera eficiente tareas de muy diversas naturalezas. Este es el caso de ROS (Robot Operating System), que ha aumentado el número de implementaciones de manera exponencial en los últimos años, convirtiéndose en todo un referente. Además, cuenta con una comunidad desarrolladora muy extensa. En este capítulo se pretende profundizar en el concepto de ROS, en su funcionamiento, su estructura y aplicaciones al proyecto.

4.1. Introducción

ROS [20], acrónimo de *Robot Opertaing System* es un sistema operativo de código abierto para robots. Provee de manera eficiente los servicios que se esperan de un sistema operativo tales como abstracción hardware, control de dispositivos a bajo nivel, implementaciones de funcionalidades comúnmente usadas, paso de mensajes entre procesos y gestión de paquetes. también incorpora herramientas y librerías para obtener, compilar, escribir y ejecutar código entre múltiples computadores. ROS es similar en algunos aspectos a otros sistemas operativos para robots como son Player, YARP, Orocos, Carmen, Orca ....

Entre los objetivos principales de ROS destacamos que fue creado con la idea de reutilizar todo el desarrollo e investigación en robótica. ROS es un sistema distribuido que permite que los programas sean diseñados individualmente y que no sean pisados unos con otros en ejecución. Estos programas están agrupados en paquetes, que pueden ser fácilmente compartidos y distribuidos. ROS también soporta código de repositorios, que habilita a que la colaboración sea también distribuida. El diseño completo de ROS desde los ficheros de sistema hasta el nivel de comunidad permite la manipulación individual sobre desarrollo e implementaciones, pero todo ello se puede hacer a la vez con las herramientas de infraestructura que trae.

A continuación se describen otras características de ROS:

- Liviano: ROS está diseñado para ser lo más ligero posible, esto es para que el código sea fácilmente transportable a otros sistemas o marcos de trabajo. El corolario de este punto sería que ROS es fácil de integrar con otros software de robots, de hecho, ROS ya ha sido implementado con Open Rave, Orocos y Player.

- Librerías transparentes: el modelo de desarrollo preferido es escribir librerías transparentes con interfaces limpias.

- Independencia del lenguaje: El marco de trabajo de ROS permite el desarrollo de programas en diferentes lenguajes de programación . Por el momento están hechas las implementaciones para Python, C++ y Lisp, además se tienen librerías experimentales para Java y Lua.

- Escalado: ROS es bastante adecuado para grandes funciones o para largos procesos de desarrollo.

Actualmente ROS solo es compatible con plataformas basadas en Linux. El software desarrollado para Ros está principalmente probado sobre distribuciones de Ubuntu, y plataformas Mac OS X. Aunque la comunidad de ROS ha contribuido al soporte para Fedora, Gentoo, Arch Linux y otras plataformas Linux. Aunque se sabe que ROS podría ser compatible con Microsoft Windows, esta opción no ha sido totalmente explorada.

4.2. Versión de ROS, instalación y entorno de desarrollo

Hay una gran cantidad de versiones en ROS como Ros Box, Ros C, Ros Diamond-Back, Ros Electric, Ros Fuerte, Ros Groovy, Ros Hydro, Ros Indigo, Jade o Kame, pero difícilmente compatibles entre sí. Actualmente, las versiones Indigo Jade y Kinetic están recomendadas por ser recientes pero a la vez estables y desarrolladas por la comunidad. Por otra parte, hay que tener cuidado puesto que cada versión funciona con unas versiones de linux determinadas, esto se puede consultar en [20].

En nuestro caso, Sacarino II trabaja con ROS Indigo, eso implica que la versión de Ubuntu a usar es la 14.04.

4.2.1. Instalación

Lo primer que tendremos que hacer será instalar el entorno de ROS, esto es mediante:

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

A continuación, añadiremos ciertas variables de sistema necesarias para el correcto funcionamiento de ROS:

wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - | sudo apt-key add -

Una vez establecido el entorno, procedemos a instalar ROS propiamente dicho:

sudo apt-get update sudo apt-get install ros-indigo-desktop-full

Seguidamente, para instalar todos los comandos relacionados con el nodo maestro y topics, haremos:

source /opt/ros/hydro/setup.bash

Con esto ya tendríamos instalado ROS, pero faltaría instalar un compilador.

4.2.2. Creación de un espacio de trabajo para ROS con Catkin

Para compilar paquetes o módulos es necesario un compilador, en esencia existen dos herramientas para hacerlo, Catkin o Rosbuild. Como en la página oficial se recomienda Catkin, es el que se ha instalado. Catkin incluye las siguientes dependencias:

- Cmake
- Python
- Gtest
- GNU C++ compiler

Se deben insertar los siguientes comandos por consola:

mkdir -p /catkin_ws/src

cd $\sim$ $/catkin\_ws/src$

catkin_initspace

cd $\sim$ $/catkin\_ws/$

catkin_make

source devel/setup.bash

Por último necesitamos referenciar nuestro espacio de trabajo local a $\sim$/.bashrc, sino hiciéramos esto, entonces herramientas como roslaunch o rosrun no serían capaces de encontrar los paquetes creados en nuestro directorio.

source /home/insert-your-username/catkin_ ws/devel/setup.bash

Con esto tendríamos la instalación finalizada. Deberíamos ser capaces de ejecutar los siguientes comandos por terminal:

- roscore $\rightarrow$ nodo maestro
- rviz $\rightarrow$ visualización
- rqt $\rightarrow$ QT Plugin Manager
- wstool

- rosdep → dependency handling tool
- rosrun → ejecutar programas
- roslaunch → lanzar ficheros .launch
- rostopic → topics actualmente en ejecución
- rosnode → nodos actualmente en ejecución
- rosbag → topics grabados
- roscd → cambiar el directoria al paquete
- rosclean → comprobar el espacio usado del fichero .log
- roswtf → resolución de problemas

El funcionamiento de algunos comandos los veremos en un apartado posterior.

4.3. Fundamentos básicos de ROS: módulos, nodos y topics

Una vez nos encontramos en un espacio de trabajo creado en catkin, como se explicaba en el apartado anterior, lo primero que nos encontramos son diversos paquetes o módulos dentro la carpeta «src» de nuestro espacio de trabajo. A su vez, estos módulos comprenden habitualmente varios programas en lenguajes compatibles con ROS como son .cpp y .py. Bien, pues estos programas ejecutables son conocidos como *nodos*, y se ejecutan con el comando «rosrun nombre-paquete nombre-nodo». El nodo es la unidad básica tratamiento de información en ROS.

Dichos nodos pueden comunicarse entre sí a través de lo que denominamos *topics*. Como cabe esperar, a cada topic le corresponde un tipo de mensaje, así por ejemplo, podemos tener topics que trabajen cadenas, enteros, estructuras, vectores,...

Para coordinar y sincronizar todos los nodos, sus topics etc, existe un nodo maestro que se puede ejecutar con el comando por consola «roscore».

4.3.1. Paradigma editor-suscriptor

En general, en el mundo de la comunicación entre procesos existen varios modelos de comunicación que, en función de la aplicación unos funcionan mejor que otros. Pongamos el clásico ejemplo de un paradigma cliente - servidor, en este caso, el cliente solicita información al servidor y éste se la devuelve, cuando acaba de abastecer a un cliente, entonces se repite el proceso con cliente2 y así. Como se puede observar, para que un cliente sea atendido el servidor tiene que estar libre en ese momento, sino tendrá que, en el mejor de los casos, esperar en una cola. De esta manera podemos decir que el cliente *tiene que esperar a la respuesta del servidor.*

En ROS hay diversos modelos implementados (topics, servicios, broadcasts...), pero el básico es en una arquitectura *editor-suscriptor*. En este paradigma el editor publica información y es el suscriptor el que, si le interesa, se puede apuntar a recogerla o no. Esto tiene como principal ventaja que el suscriptor no tiene que esperar a ninguna respuesta del editor, sino que el editor va publicando su información y es el suscriptor quien decide si tomarla (suscribirse al topic) o no. Un ejemplo de este paradigma se encuentra en la figura 4.1.

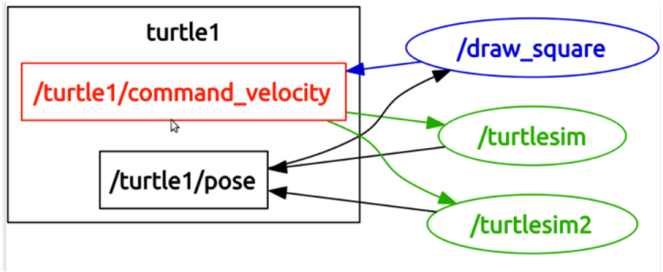


Figura 4.1: Modelo Editor Suscriptor

En la imagen 4.1 presentada, habría dos topics, uno sería /turtle1/command_velocity y otro sería /turtle1/pose. El nodo /draw_square sería editor del topic /turtle1/command _velocity y suscriptor del topic /turtle1/pose. El nodo turtlesim sería suscriptor del topic /turtle1/command_velocity y editor del topic /turtle1/pose, al igual que el nodo /turtlesim2. A cada uno de los topics mencionados le correspondería un tipo de mensaje, por ejemplo en el caso del topic /turtle1/command_velocity, el mensaje sería del

tipo Twist, que se encuentra definido en la librería std_msgs y tiene como campos velocidades angulares y lineales. El nombre de turtle1 es el espacio de nombres de ambos topics, es por eso que ambos comienzan por /turtle1/. Conviene aclarar que, en general, puede haber más de un nodo suscrito a un topic y más de un editor en un topic.

4.3.2. Servicios en ROS

Aunque el modelo editor subscriptor es en el que se fundamentan los topics, también existen los *servicios* en ROS. Básicamente los servicios funcionan como los topics, pero se diferencian en que usan una comunicación cliente-servidor, es decir, cuando el servidor ha procesado la petición (request) del cliente, entonces manda una respuesta (response) al cliente confirmándole si todo ha ido bien o no. Un ejemplo de uso se verá mas adelante, cuando se hable del servicio que mueve los servomotores situados en la cabeza del robot Sacarino II.

4.4. Librerías y Plug-ins adicionales

En la instalaciones de ROS se incluyen una serie de herramientas que conviene, al menos, comentar su funcionamiento, puesto que se han utilizado en la realización del proyecto. Además, a continuación se describe el uso del paquete de repositorios OpenNI.

4.4.1. OpenNi, Transformed Frames y modelo Broadcast

Con la idea de programar a alto nivel, se instalaron las librerías OpenNi[1] que trabajan con los datos de salida de la kinect para transformarlos en nubes de puntos, imágenes de disparidad y otras formas de información para el post-procesado y post-tratamiento. En realidad, son unas librerías que fueron sacadas para la distribución Ros-Hydro, pero, al haber sido compiladas manualmente, han servido para ROS-Indigo. Junto a ellas, se han instalado los drivers para Kinect «Nite».

Otra herramienta que aprovecha el potencial de openni es *openni_tracker*, la cual, a

[1]Su instalación puede ser encontrada en [21]

partir de los topics publicados por el resto de nodos de OpenNi, en base a matrices de transformación (En adelante *transformed frames*) y de la imagen RGB de la cámara es capaz de publicar sistemas de referencia sobre las articulaciones de una persona que se encuentre en ese momento en el campo visual de Kinect. Esta es una herramienta muy potente y que va a ser esencial a la hora de plantear un modo de abordar en primer objetivo fijado en este proyecto.

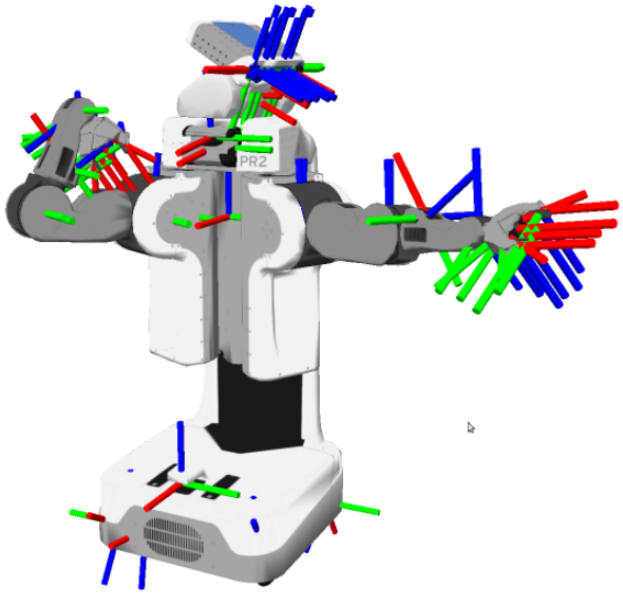


Figura 4.2: Transformed Frames marcados sobre un robot

El paradigma de comunicación de los *transformed frames* es conocido como broadcast, uno a todos. Donde un topic publica a todos los nodos su información.

4.4.2. Otras herramientas

- Rviz: es un potente programa de visualización de información muy variada. Podemos ver desde la imagen de la cámara de Kinect a la nube de puntos que publica. Tiene una interfaz sencilla e intuitiva. Sobre todo, su uso se ha enfocado hacia la visualización de los transformed frames que publica openni_tracker en tres dimensiones.

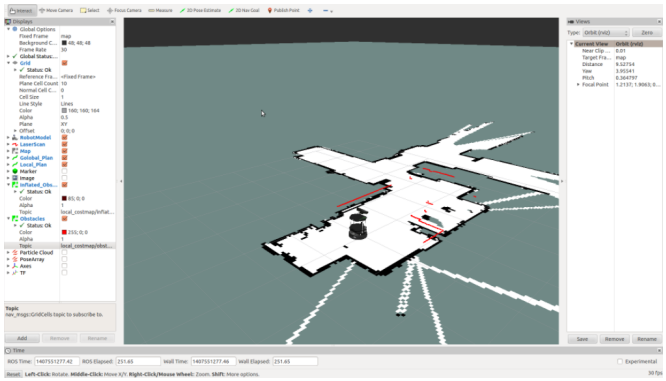


Figura 4.3: Rviz

- Rqt: se trata de una interfaz que proporciona al usuario diversos servicios entre los que se encuentra el poder enviar mensajes a los topics de manera manual, esto ha sido especialmente útil a la hora de depurar el funcionamiento de los nodos programados.

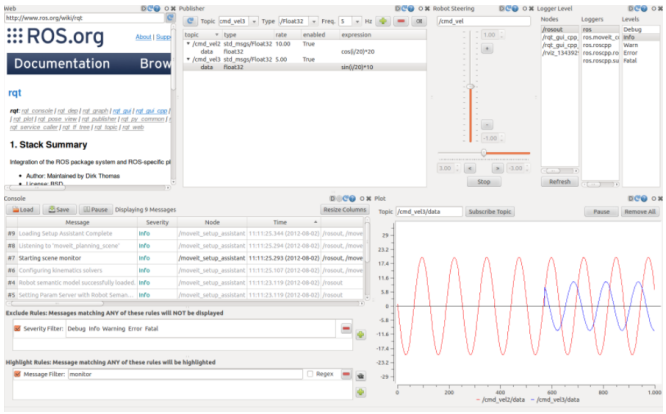


Figura 4.4: Rqt

- Rqt_graph: es un comando que se ejecuta por consola y que sirve para ver, mediante un grafo, los nodos en ejecución en ese momento, topics y espacios de nombres y cómo están relacionados entre sí.

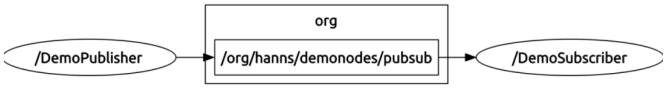


Figura 4.5: Rqt_graph

- Turtlesim: se trata de un conjunto de programas tutoriales que introducen al usuario al uso de los *transformed frames*. Ha sido de gran utilidad durante el aprendizaje. Básicamente se trata de dos tortugas en una ventana 2D, una que controla el usuario con las flechas y otra que sigue a la primera. Por lo que da una idea aproximada de cómo insertar el control en lazo cerrado con los tf.

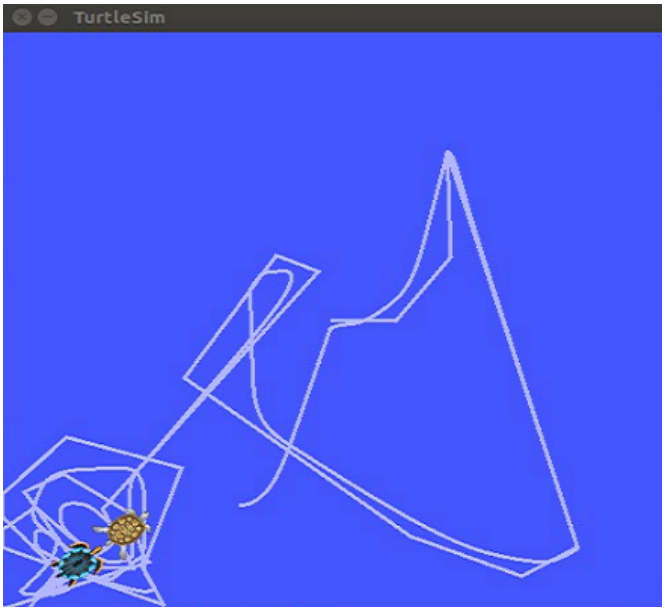


Figura 4.6: Turtlesim

- tf_view_frames: comando que se introduce por consola que permite ver, a través de un grafo, el árbol formado por los transformed frames en ese momento.

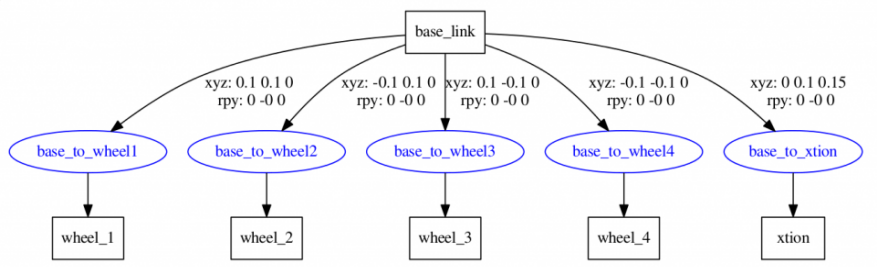


Figura 4.7: Tf_view_frames

Capítulo 5

SUBSISTEMA DE SEGUIMIENTO

En este capítulo se comienza la segunda parte del presente proyecto, donde se exponen los programas, funcionalidades implementadas y utilidades para cumplir con las especificaciones iniciales. Se parte de los conceptos introducidos en capítulos anteriores tales como Kinect, nube de puntos, *Transformed frames*, ROS, OpenNi, robótica social, Sacarino II, y otros. En concreto, este capítulo está dedicado al sistema de seguimiento (*tracking*) con el que se va a satisfacer el primero de los objetivos marcados para lograr una mejor interacción hombre-máquina.

5.1. Planteamiento. Sistemas de referencia

Uno de los comportamientos sociales más aceptados es el de mirar a los ojos a la persona con la que se está interactuando. El problema se puede abordar de diversas maneras. Primeramente, se van a repasan los elementos de sensorización con los que se cuenta:

- Sensor de visión 3D Microsoft Kinect.
- Sensor de visión 2D láser SICK LMS100.

Se trata ahora de combinar el uso de estos dos elementos. Por un lado, cuando el robot exprese alguna emoción (se trata en profundidad en el capítulo 6) se va a usar el láser como sistema de seguimiento (solamente seguimiento en el plano panorámico), la

justificación radica en que si se usara Kinect y se pusiera un gesto que implicase bajar el cuello a una intensidad marcada, al estar Kinect ubicada sobre la cabeza del robot, el usuario quedaría fuera del campo visual, perdiendo el seguimiento de este.

Por otro lado, cuando el robot mantenga la expresión neutra, Kinect va a ser quien realice el seguimiento, esta vez en ambos grados de libertad.

5.1.1. Introducción al seguimiento con Kinect

Los *Transformed Frames* que publica openni_tracker visualizados en Rviz se muestran en la figura 5.1. Como se puede observar, se etiquetan cada una de las articulaciones del usuario en el campo visual p.ej: head_3,neck_3,etc. Esto es posible debido a que, por un lado, con la imagen color, internamente hace un proceso de Esqueletización, que consiste en adelgazar ciertas partes que interesan en una imagen, una vez aisladas con operadores de morfología, y por otro lado, obtiene las coordenadas de los puntos mediante inferencia de posición tras haber segmentado (algoritmo *mean shift*) . Cabe decir que cuando openni_tracker pierde a un usuario, entonces cuando vuelva a detectar a otro, lo etiqueta variando el número del final, p.ej head_1, head_2 ....

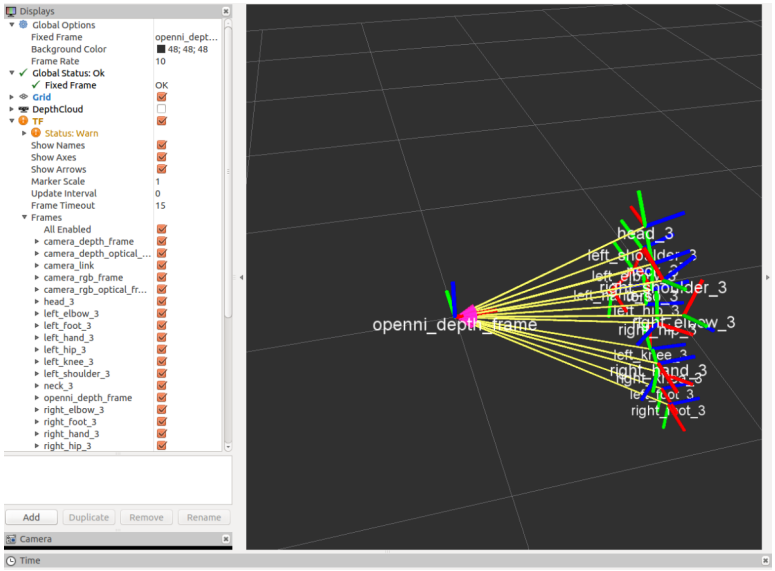
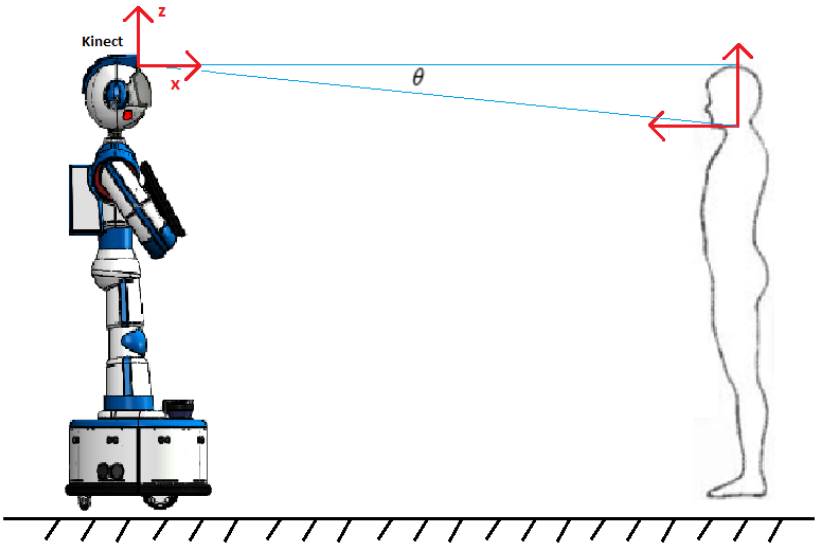


Figura 5.1: *Tranformed Frames* visualizados con Rviz

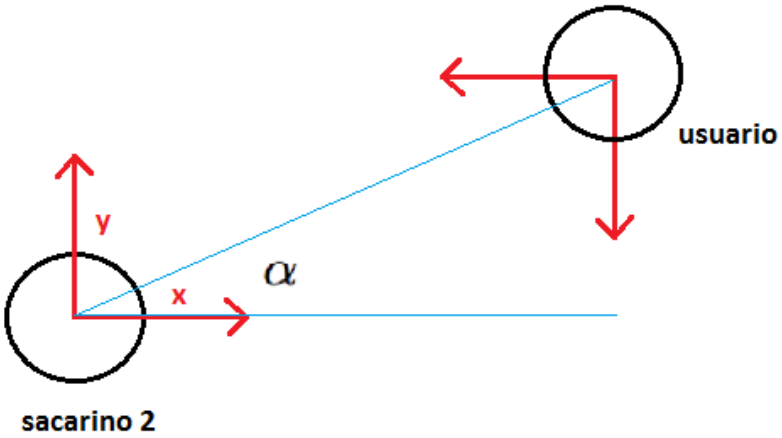
De todos los sistemas de referencia que publica openni, interesan exclusivamente los

sistemas de referencia del cuello, por dos razones, una de ellas sería porque al seguir al cuello del usuario la kinect, los ojos quedan más a la altura del usuario, y la otra es porque es más difícil que el cuello se salga fuera del alcance del rango visual de kinect, en cuyo caso se podrían producir inconsistencias que hicieran disminuir la robustez del conjunto.

Sea un usuario que se encuentre en frente de Sacarino II, como se muestra en la figura 5.2.



(a) Vista de perfil



(b) Vista cenital

Figura 5.2: Vistas de la interacción

Se denotan los ángulos que forman los sistemas de referencia usuario - kinect como tita y alpha, que representan el error entre la posición actual de los servomotores y la que deberían estar para que el robot fijase su mirada en el usuario. A su vez, x será la distancia del origen del sistema usuario al origen del sistema kinect, medida a lo largo del eje x del sistema kinect. De manera análoga se definen y y z. Con el problema ya planteado y la información de los sistemas de referencia colocados de esta manera, se procede a la explicación de las diversas estrategias de seguimiento con Kinect.

5.2. Estrategias de seguimiento con Microsoft Kinect

A continuación se presentan diferentes aproximaciones para resolver el problema de seguimiento con Kinect.

5.2.1. Estrategia mediante modulación de la referencia de velocidad

Esta primera estrategia se basa en la ejecución asíncrona de un conjunto compuesto por cuatro instrucciones «subir cuello»,«bajar cuello»,«cuello izquierda», «cuello derecha». Donde el control se realiza exclusivamente por velocidad.

Dependiendo del signo que presenten los ángulos α y θ, quedan definidos los cuatro cuadrantes que dan lugar a la ejecución de cada una de las instrucciones del conjunto. De esta manera, por ejemplo, un α mayor que cero significará mover el cuello a la izquierda y un θ mayor que cero, mover el cuello hacia abajo.

Para el movimiento de un servomotor se requieren, típicamente, definir posición y velocidad. La posición viene fijada por la instrucción que se está ejecutando, de tal forma que cuando se tenga un α menor que cero, cuello a la derecha, entonces la referencia de posición sería la máxima posición mecánicamente aceptable a la que puede llegar ese servomotor, en el caso de Sacarino II, +1,5 [rad]. Y, viceversa, un α mayor que cero, cuello a la izquierda, implica una referencia de posición al servomotor de -1.5 [rad], es decir, totalmente a la izquierda. El mismo tratamiento se puede llevar a cabo con θ.

La segunda variable que se ha de establecer es la velocidad, este es un punto crítico

del algoritmo, pero antes se deben aclarar los siguientes puntos:

- En general, las tarjetas controladoras de servomotores suelen dar un significado especial al comando de «velocidad 0». En el caso de la tarjeta SSC32 controller de Lynxmotion una referencia de velocidad 0 se traduce como la velocidad máxima que puede soportar ese servomotor. Debido a esto, en la programación del algoritmo se ha definido, como referencia cero, una velocidad del orden de centésimas, para evitar que la orden de cero signifique todo lo contrario a lo que se pretende, que es hacer parar el servomotor.
- Se asume la hipótesis de que la dinámica del sistema es lo suficientemente rápida como para reaccionar ante cambios imprevisibles del usuario.
- Se asume que el algoritmo es capaz de ejecutarse a una frecuencia suficiente como para hacer cumplir el teorema de Shannon. En el caso de este estudio, se toma como frecuencia natural del sistema 1 segundo (el usuario puede experimentar un cambio sustancial de posición a lo largo de un segundo). Con lo que, para garantizar la reproductibilidad de la señal, el teorema obliga a tomar una frecuencia de muestreo de, al menos, 0.5 segundos (2 Hz). Sin embargo, en la práctica se suele tomar de 8 a 10 veces la frecuencia natural, por lo que se debe garantizar un período de muestreo de, al menos, 0.1 s.

.

Aclarados estos puntos, la velocidad se controla mediante el módulo de los ángulos α y θ. Para ello se ha definido una función lineal dependiente del módulo de estos ángulos, de tal forma que cuando éste sea pequeño, la velocidad sea pequeña y, cuando el módulo sea grande la velocidad sea mayor.

$$g(x) = \begin{cases} -2 \cdot -x & si \quad x < 0 \\ 2 \cdot x & si \quad x > 0 \end{cases} \tag{5.1}$$

Para no incurrir en oscilaciones se ha incluido un intervalo de tolerancia [0 rad a 0.1 rad] en el que, si el módulo del error se encuentra dentro de este umbral, entonces la referencia de velocidad que se envía es de velocidad cero [1].

[1] Tener en cuenta que nuestra referencia de velocidad cero es una velocidad pequeña del orden de

Por tanto queda definida la estrategia, con las suposiciones planteadas. El pseudo-código simplificado del algoritmo sería el que se presenta a continuación:

```
Inicializar variables posición y velocidad de los servos
  Calcular_alpha
  Calcular_tita
   Si alpha >0:
     posición_servo_0 := -1.5
   Sino:
     posición_servo_0 := 1.5
   Fin Si
   Si tita >0:
     posición_servo_1 := -1.5
   Sino:
     posición_servo_1 := 1.5
   Fin Si
   velocidad_servo_0 := Calcular_Velocidad(alpha)
   velocidad_servo_1 := Calcular_Velocidad(tita)
   Si abs(alpha) <0.1:
     velocidad_servo_0 := ZERO
   Fin Si
   Si abs(tita) <0.1:
     velocidad_servo_1 := ZERO
   Fin Si
   Mover_Servo_0(posición_servo_0,velocidad_servo_0)
   Mover_Servo_1(posición_servo_1,velocidad_servo_1)
```

centésimas, en ningún caso cero

5.2.2. Estrategias mediante filtrado de referencia de posición

Este tipo de estrategias se basan en el control de ambas, posición y velocidad, a diferencia de la estrategia vista en el apartado anterior. Con lo expuesto en 5.1.1, y teniendo en cuenta que la técnica se basa en el control de la posición de los servomotores, es sencillo deducir que para que el robot sea capaz de mirar a un usuario, *entonces los ángulos tita y alpha presentados en las subfiguras 5.2 han de hacerse 0.* A nivel software, se tiene información del punto origen del sistema de referencia cuello respecto a la kinect, por lo que esos ángulos, en cada instante, se hallarían de la siguiente manera:

$$\theta = atan(\frac{z}{x}) \tag{5.2}$$

y, de manera análoga podríamos hallar:

$$\alpha = atan(\frac{y}{x}) \tag{5.3}$$

Con (5.2) y (5.3) en [rad]. [2] Otro método alternativo al cálculo de los ángulos α y θ es el método de los cuaternios (una extensión de los números reales, similar a la de los números complejos), los cuáles podemos transformar en los tres ángulos fundamentales RPY(Roll, Pitch y Yaw), Guiñada, cabeceo y alabeo.

Esquema de control

En cada instante de muestreo se calcula el valor de ambos ángulos (α y θ) y se manda una señal de control a los servomotores en función de los valores de los ángulos en ese instante. Dos parámetros básicos que requieren los servomotores para su movimiento son velocidad y posición.

Todo esto conforma la estructura de control, que se resume en la ilustración 5.3. Donde el Setpoint o valor de consigna sería 0, el bloque *f(e)* varía según la estrategia implementada, la planta serían los servomotores, y, la salida, la nueva ubicación del usuario respecto de la Kinect.

[2] En la biblioteca math de C++ se recomienda el uso de atan2, para considerar cambios de cuadrante.

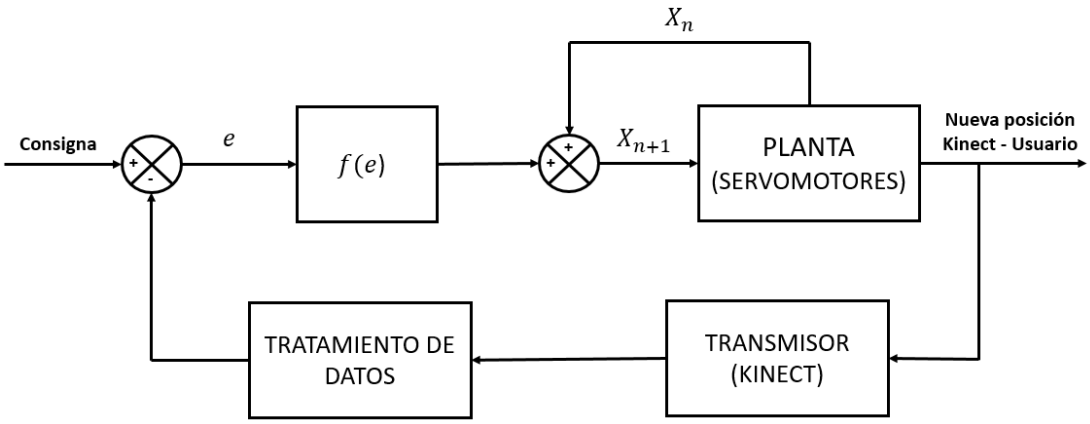


Figura 5.3: Esquema básico de control implementado en seguimiento facial

Tratamiento lineal del error

Uno de los problemas que se dan a la hora de hacer seguimiento, es el movimiento brusco o a trompicones de la cabeza del robot. Esto tiene un impacto muy negativo sobre la interacción hombre-máquina, por tanto es un problema que se debe evitar. Para abordar esto se introduce en 5.3 el bloque de tratamiento del error *f(e)*, donde se programa una función que, en el caso de la presente estrategia, se trata de una función lineal a trozos.

Por una parte, interesa filtrar la próxima referencia que se envía al servomotor. Así, si en un instante de tiempo se tiene que alpha (error) vale 0.3 [rad] y la posición actual del servomotor es 0.1 [rad], en vez de enviar la próxima referencia de 0.4 [rad] se envía la posición actual más una proporción del error, dando lugar a la siguiente fórmula:

$$X_{n+1} = X_n \pm e \cdot f(e) \quad conf(e) \in (0,1), e \in (0, e_{max}) \tag{5.4}$$

siendo *Xn* el ángulo del servomotor en el instante de tiempo actual, *Xn+1* el ángulo del servomotor en el próximo instante de tiempo, e el error, e_{max} el error máximo admisible y donde la función *f(e)* se relaciona con el error a través de la función lineal a trozos definida por[3] (5.5) y representada en 5.4.

[3]Se detalla solamente la parte positiva al ser simétrica

$$f(e) = \begin{cases} 2{,}5 \cdot e & si \quad 0 < e \leq 0{,}2 \\ (0{,}833 \cdot e) + 0{,}33 & si \quad 0{,}2 < e < 0{,}5 \\ 0{,}85 & si \quad e \geq 0{,}5 \end{cases} \tag{5.5}$$

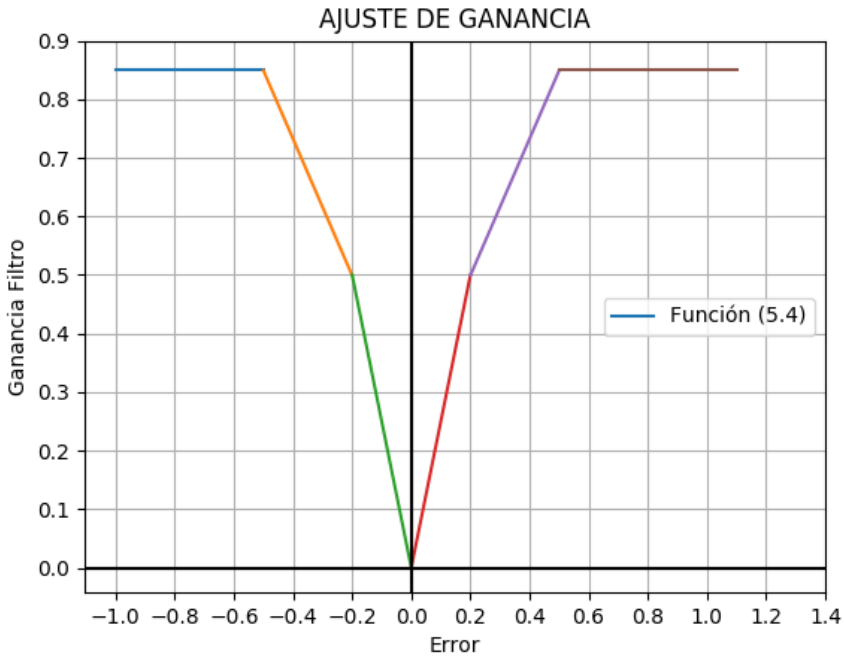


Figura 5.4: Curva de ajuste de ganancia

Por otra parte, la velocidad se planifica a través de otra curva experimental lineal a trozos (5.6). Cuando un usuario está lejos de la mirada del robot, el error es grande, por lo que se manda una referencia de velocidad mayor, que cuando ocurre lo contrario 5.5.

$$f'(e) = \begin{cases} 0{,}3 & si \quad 0 < e \leq 0{,}6 \\ (0{,}166 \cdot e) + 0{,}2 & si \quad 0{,}6 < e < 1{,}2 \\ (0{,}2 \cdot e) + 0{,}166 & si \quad 1{,}2 < e < 1{,}7 \\ 0{,}5 & si \quad 1{,}7 < e \leq 2 \end{cases} \tag{5.6}$$

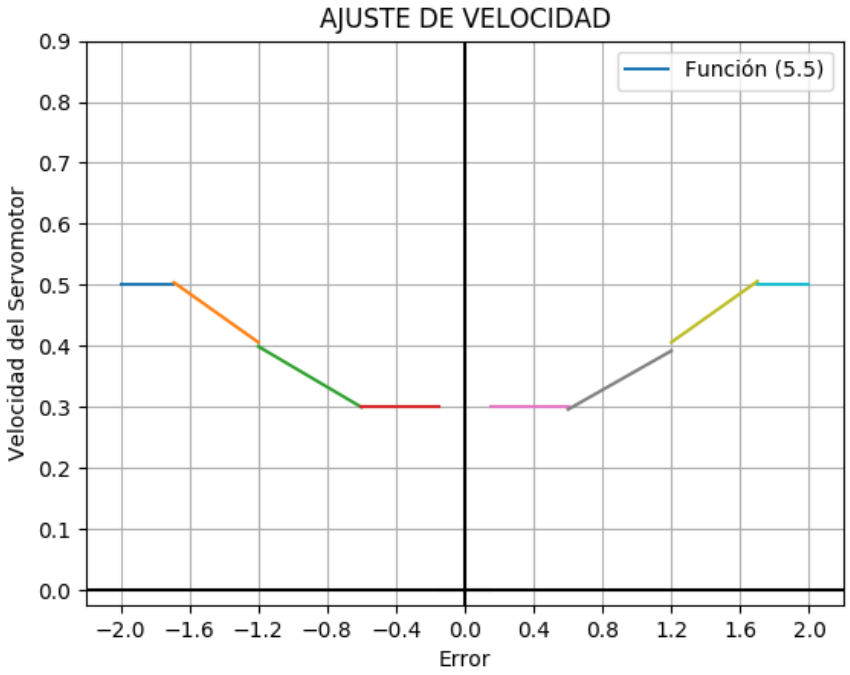


Figura 5.5: Curva de ajuste de velocidad

De esta manera se pretende que ante un salto o rampa (movimiento brusco o más suave del usuario respectivamente) el sistema vaya a la referencia de manera suave pero fluida teniendo en cuenta el módulo del error en cada momento.

Tratamiento no lineal del error

En general se puede decir que los movimientos que un usuario puede hacer delante del robot son imprevisibles. Esto significa que la naturaleza de la variable controlada es no lineal y por tanto, sino se llevan los esfuerzos de control a zonas y transiciones suaves de trabajo, puede desembocar en una disminución de la vida de los servomotores (actuadores), además, de no hacerse esto, el sistema puede resultar oscilante, o, ante cambios muy bruscos, inestable.

La estrategia que se expone en el presente apartado tiene como objetivo no solamente que el seguimiento sea fluido y continuo, sino que además los esfuerzos de control de posición y la velocidad de cambio en las referencias a los servomotores sean asumibles y no oscilatorias ni discontinuas.

Esta técnica consiste en tratar la señal de error mediante una función no lineal (tipo sigmoide), de forma que cuando el error sea pequeño, los esfuerzos de control sean muy pequeños y cuando el error aumente, los esfuerzos de control aumenten en mayor medida.

Es bastante común en inteligencia artificial el uso de la función sigmoide, más concretamente en redes neuronales como función de activación de una determinada neurona. Se puede construir a partir de funciones elementales, típicamente mediante la función exponencial o la familia de razones hiperbólicas. En el caso de esta estrategia, se ha considerado la sigmoide en la forma (5.7).

$$f(e) = \frac{e_{max}}{1 + exp(e_{max} - 2e)} \tag{5.7}$$

Donde el rango de valores de *f(e)* es de [0,e_{max}] para valores de e en [0,e_{max}]. En la figura 5.6 se representa dicha función.

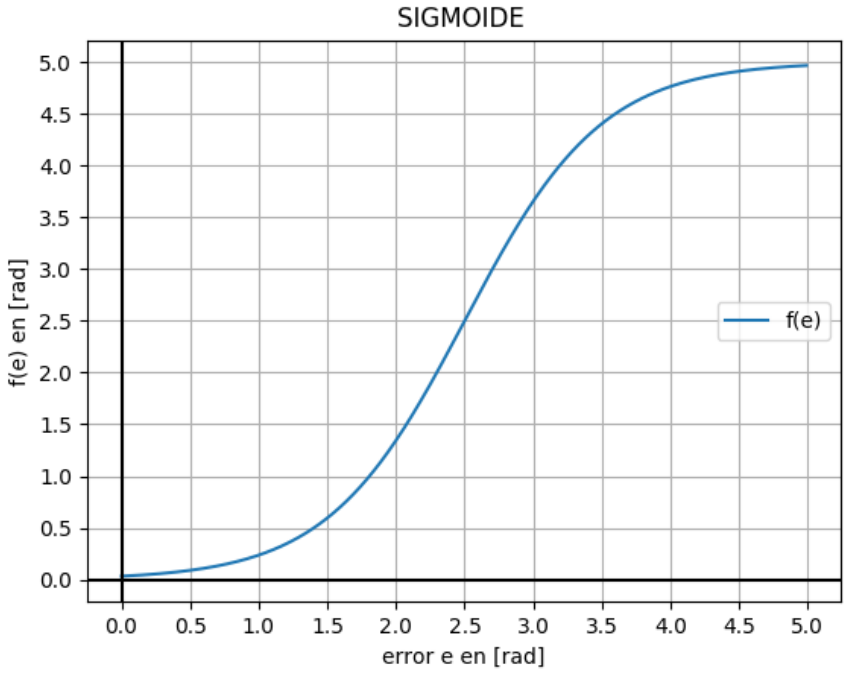


Figura 5.6: Curva de tratamiento no lineal para $e_{max} = 5$

Considerando este tratamiento del error, la ecuación que calcula la próxima referen-

cia al servomotor sería la presentada en (5.8).

$$X_{n+1} = X_n \pm \frac{e_{max}}{1 + exp(e_{max} - 2e)} \tag{5.8}$$

Junto con la posición se debe de especificar la velocidad con la que se desea llegar a cada nueva posición, en este sentido, la referencia de velocidad que se envía a la tarjeta controladora es la máxima admitida para cada servomotor, pues la tarjeta controladora se encarga de generar los perfiles de aceleración y deceleración (figura 5.7).

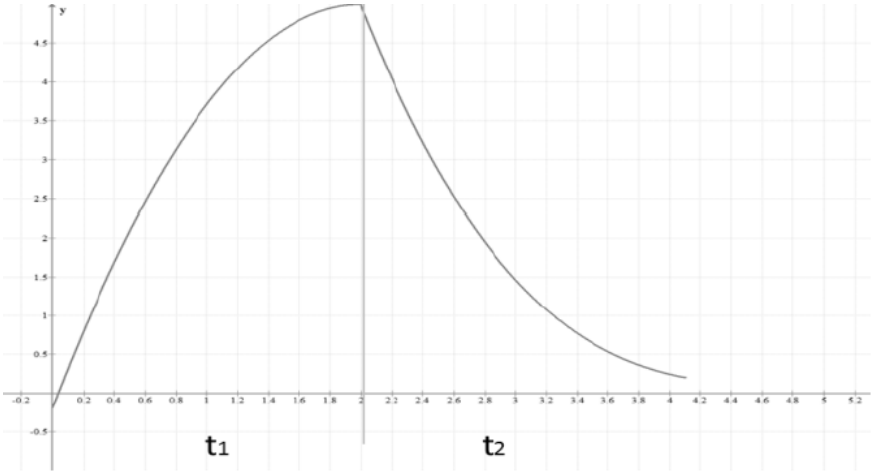


Figura 5.7: Curva de aceleración/deceleración del servomotor [6]

5.3. Seguimiento con el láser SICK

5.3.1. Introducción

El láser SICK LMS100[4] como se explica en el capítulo 2, está ubicado en la base, junto a los pies del robot.

El principio de operación del láser se basa en una medición por tiempo de vuelo. El láser envía pulsos cortos de luz a la vez que un reloj electrónico es puesto en marcha. Cuando la luz impacta sobre un objeto, es reflejada y recibida por el sensor, parando el reloj. De esta forma, a través del tiempo entre envío y recepción, es capaz de calcular la distancia al objeto.

[4]La hoja de características se puede encontrar en el apéndice C de este proyecto

Además, internamente contiene un espejo que rota a velocidad constante que deflecta los pulsos lumínicos para lograr un arco de 270º con una resolución de entre 0.25º y 0.5º. el primer rayo comienza en -45º relativos a la espalda del sensor.

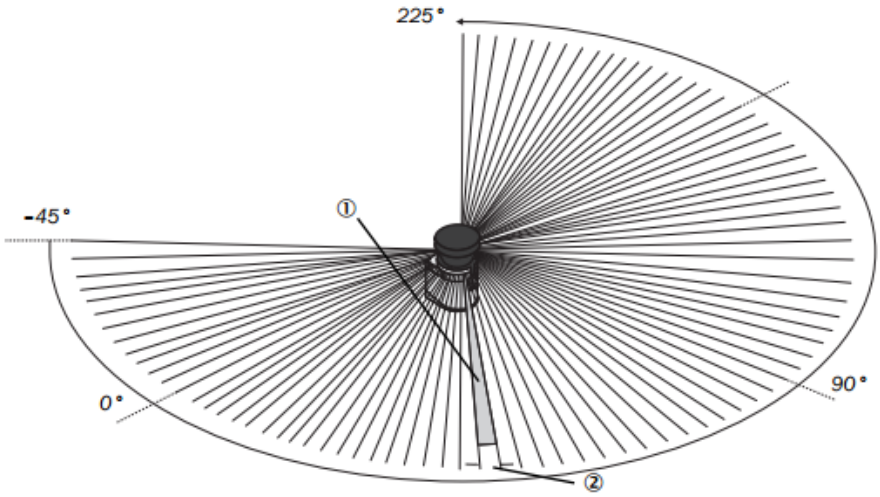


Figura 5.8: Láser SICK LMS100

El objetivo del láser es proporcionar un seguimiento de la cabeza del robot al usuario en el plano panorámico mediante la detección de pares de piernas.

5.3.2. Estrategia de seguimiento

En primer lugar es necesario tratar la información que ofrece el láser y acondicionarla para poder trabajar con ella a alto nivel. Toda la información de las distancias obtenidas por el láser son recogidas por un primer programa en ROS llamado «Sacarino Láser» cuya función es convertir esos datos a nubes de puntos manejables en ROS. A continuación, se define el sistema de referencia «base» con respecto al cual se referencian las coordenadas de cada punto de la nube.

Una vez se ha terminado la etapa de acondicionamiento, comenzaría la siguiente fase, donde entran en juego los algoritmos de detección y seguimiento. Por una parte, el algoritmo de detección programado en un conjunto de nodos ROS llamado *leg_detector* [22], se encarga de, mediante un método probabilista, de detectar agrupaciones de puntos (*clusters*) que puedan corresponderse con pares de piernas y, por ende, de usuarios.

Leg_detector ofrece como salida todas las posibles personas detectadas en un vector de objetos de tipo *PositionMeasurement*, cuyos campos son los siguientes:

- *Header*: se trata de una estructura compuesta por un identificador (*id*) y un campo de tiempo (*stamp*).

- *Name*: nombre que recibe la persona en concreto. Típicamente suele tratarse de la cadena de caracteres «people» seguido de un número que varía de una persona a otra.

- *Object_id*: identificador que recibe el objeto dentro del vector. Formado por la unión de dos cadenas de caracteres asociadas a los pares de piernas que conforman esa persona. Por ejemplo: «legtracker121|legtracker457».

- *Pos*: campo de tipo *geometry_msgs/Point* que contiene las coordenadas x,y de la persona detectada (punto medio del par de piernas).

- *Reliability*: o fiabilidad, da muestra de cuán seguro está el algoritmo de que verdaderamente la nube de puntos se corresponde a una persona. Varía de 0 a 1.

- *Covariance*: Muestra la covarianza (relación) que guardan el movimiento de una pierna con la otra. Si se trata de las piernas de la misma persona, entonces debería ser un número grande y viceversa.

- *Initialization*: campo de tipo byte de inicialización.

En la imagen 5.9, se ilustra un conjunto de personas detectadas con el láser y visualizadas con Rviz. Como se puede observar, en rojo se muestran las piernas y en el globo verde el punto medio entre ellas (ubicación de la persona), en este caso, se estaban detectando dos personas. Sin embargo, en realidad solamente había una persona, tratándose de un falso positivo en el caso de la otra. Es por ello que uno de los campos que más interesan a la hora de programar el nodo de seguimiento, es el de fiabilidad, puesto que es de vital importancia a la hora de evitar el seguimiento de ruido u otros objetos que no sean personas.

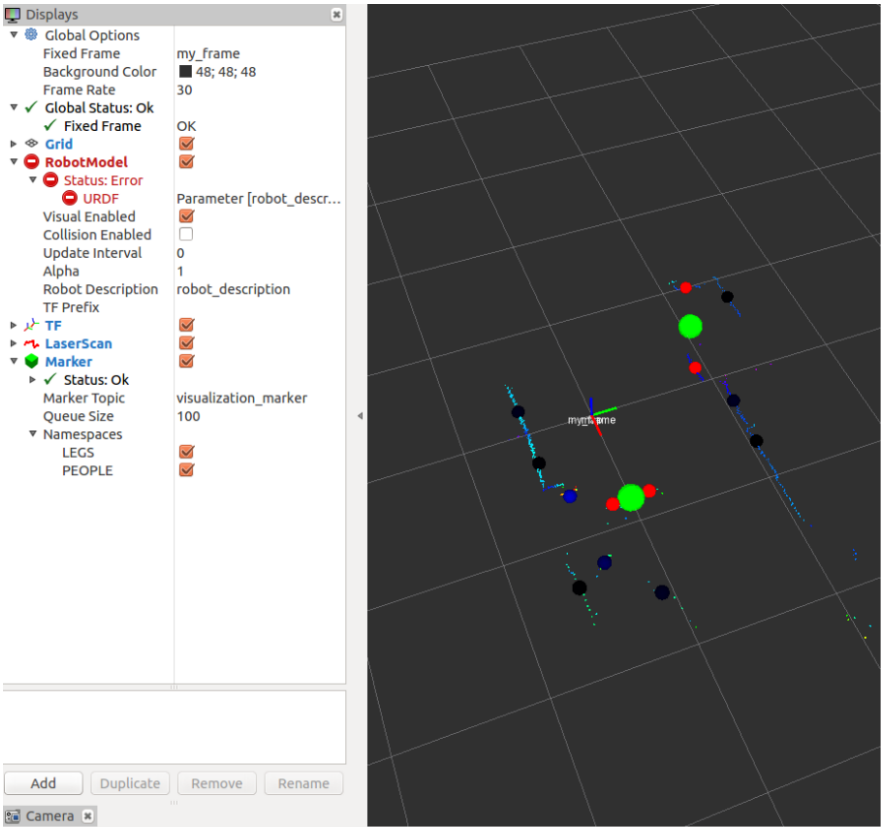


Figura 5.9: Personas detectadas, visualizadas con Rviz

La estrategia de seguimiento consiste en ordenar el vector de posibles personas de mayor a menor de acuerdo al parámetro «fiabilidad». Una vez ordenado se toma el primer elemento y, si su fiabilidad es mayor que un umbral (0.25, 25 %) entonces se empieza a hacer seguimiento a esta persona que queda automáticamente identificada en el vector por su campo *object_id*.

A partir de este momento, el sistema de control toma nota del número de identificador de esa persona, y, en cada iteración la busca por dicha etiqueta, con el fin de realizar un seguimiento estable de la misma incluso en presencia de otras personas. Las únicas dos condiciones que se pueden dar para que el robot cambie de persona a la que hace el seguimiento son las siguientes:

1. Que la persona salga del rango de alcance del láser, en cuyo caso, se volverá a tomar del vector de entrada, la persona que tenga la fiabilidad más alta.

2. Que la persona seguida actualmente permanezca estática y que, a su vez, otra

persona varíe su posición de manera significativa, en cuyo caso se pasará a seguir a esta última.

Finalmente, para que la cabeza del robot siga al usuario es necesario enviar a la controladora posición y velocidad del servomotor. Para ello, el ángulo se calcula de manera análoga a cómo se hacía en el seguimiento en plano panorámico con Kinect. De la persona se toman, dentro del campo *pos*, las coordenadas «x» e «y» y se calcula el ángulo que se tiene que enviar a la controladora mediante:

$$\varphi = arctan(\frac{y}{x}) \tag{5.9}$$

con φ en [rad] y «x» e «y» en metros. La velocidad que se envía para llegar a la posición final es la velocidad máxima permitida al servomotor de movimiento panorámico del cuello.

5.4. Estructura de programación de nodos.

En el planteamiento a la hora de implementar sobre el lenguaje de programación Python se ha tenido en cuenta la modularidad del diseño que permite ROS. Es por lo que se han separado las distintas funcionalidades en nodos distintos. Por otro lado, se ha seguido el paradigma de programación orientada a objetos (POO).

Un concepto que puede ser de utilidad es el de arquitectura, donde cada nivel proporciona una serie de servicios al nivel superior, de tal manera que lo que ocurra por debajo sea como una «caja negra» para los niveles superiores, no importa cómo lo hagan sino que se le envían unas entradas y se ofrecen unas salidas.

Para llevar a cabo el seguimiento se tienen varios nodos que conformarían el nivel 1 en la arquitectura: un nodo llamado *tf_publisher.py* cuya función es la detección con Kinect y, a través de los *transformed frames*, publica (topic /target_point) y actualiza las coordenadas de los sistemas de referencia cuello que sean detectados a través de mensajes tipo geometry_msgs PointStamped (ver tabla

Tipo de Dato	Nombre del campo	Descripción
Header	header	Cabecera que contiene un identificador de mensaje y un tiempo asociado
Point	point	Estructura con campos x, y, z

Tabla 5.1: *Definición del mensaje PointStamped*

Otro nodo llamado *tracker.py* donde se implementa toda la lógica referente a los algoritmos de seguimiento tanto de Kinect como del láser. Por último se tendrían el conjunto de nodos ya comentados *leg_detector* [5] que se encargan del tratamiento de datos recibidos por el láser y detección de personas por medio de pares de piernas.

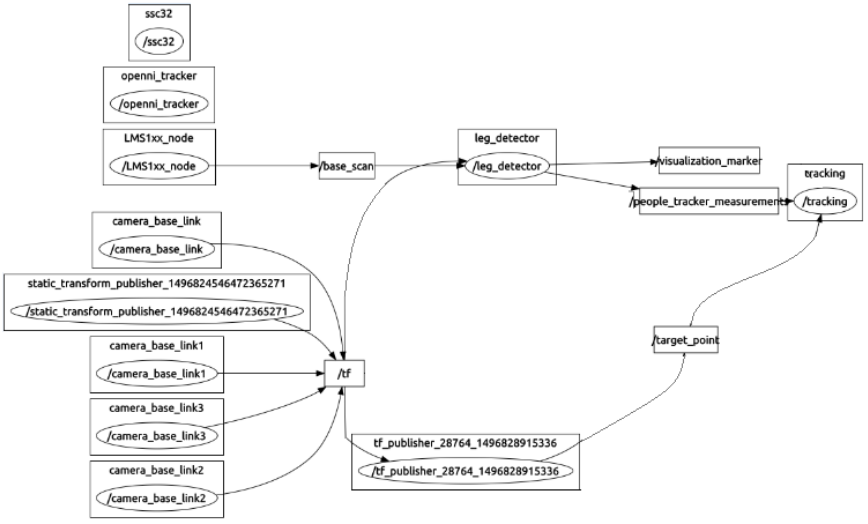


Figura 5.10: Grafo del subsistema de seguimiento visualizado con rqt_graph

Por otra parte, en el nivel 0 de la jerarquía se encontrarían las librerías de bajo nivel que se comunican con la tarjeta driver de los servomotores (lib_SSC32.py) y un servicio ROS que hace de interfaz con esa librería (ros_ssc32).

Los servicios en ROS se basan en el paradigma cliente-servidor, de forma que el cliente envía una petición al servidor, éste la procesa y envía una respuesta al cliente mediante un mensaje de reconocimiento (ACK).

En concreto, el servicio de la controladora SSC32 controller aporta una interfaz que

[5] http://wiki.ros.org/leg_detector

proporciona tres servicios fundamentales para un modelo cliente-servidor establecido por cada servomotor:

- El primero sería *move_angle*: mueve los servomotores hasta un ángulo determinado, a una velocidad determinada. Básicamente acepta tres parámetros de entrada. El primero de ellos sería el argumento *pos*, en el que se indica el ángulo al que se le quiere enviar (en magnitud absoluta). En el segundo argumento *angvel* se trata de especificar la velocidad en radianes por segundo. Finalmente, acepta un tercer parámetro llamado *endgroup* en el que se especifica si se quiere ejecutar la orden de inmedidato, o si se desea encolar la orden. Es por tanto una varibale booleana. Si se han encolado varias órdenes, en el momento en que el campo valga «True» (típicamente en la última) la controladora comenzará a ejecutar todas las órdenes de manera muy rápida, logrando sensación de paralelismo. Además, el servicio devuelve una variable boolenana si la orden ha tenido éxito.

  move_angle(pos, angvel, endgroup)

- *read_angle* sería otro de los servicios: en el que se puede consultar el ángulo en el que se encuentra el servomotor en un instante determinado. No tiene argumento de entrada y como salida devuelve un número en coma flotante con la posición angular en radianes.

  read_angle()

- El último servicio es *Ismoving*: en el que se puede comprobar si un servomotor está actualmente en movimiento. No acepta parámetros de entrada y devuelve una varibale booleana donde «True» significa estar en movimiento y «False» parado.

  Ismoving()

Capítulo 6

SUBSISTEMA DE EXPRESIÓN GESTUAL

Este capítulo desarrolla el sistema de expresiones implementado en el robot Sacarino II. El campo de la generación de expresiones por parte de los robots ha merecido particular atención en los últimos años con el crecimiento de la robótica social, en el intento de dotar a robots humanoides de una expresividad lo más parecida al ser humano. Es por ello que existen varios estándares como FACS, MAX, EMG... y que no solamente han involucrado el campo técnico sino también otras áreas de conocimiento como son psicología y animación.

6.1. Estándar de codificación facial FACS

FACS o *Face Action Coding System* es el sistema de codificación utilizado para describir los movimientos de los músculos de la cara y analizar las expresiones faciales. Esta clasificación es el resultado de las investigaciones de psicólogos como Paul Ekman, Wallace, Hager y Friesen en los años 70 [23].

Una primera edición se publicó en el año 1978 y otra actualización importante se produjo en el año 2002. El sistema de codificación facial es el método más ampliamente reconocido que se usa para codificar los movimientos faciales humanos.

La unidad básica del FACS es la unidad de acción. (UA) Una UA (o Action Unit) representa las actividades musculares que producen cambios momentáneos en las apariencias faciales. Dicho de otra manera, una unidad de acción es un código numérico para describir los movimientos de los músculos faciales. El mapeo entre las UA y los músculos faciales no es necesariamente 1-a-1; algunos UA se componen de más de un músculo, y otros UA se basan en describir movimientos separados del mismo músculo.

El acto de describir movimientos faciales utilizando el sistema de codificación facial se llama FACS de codificación. Códigos múltiples FACS se ensartan usando más ('+'). Por ejemplo 1 + 2 representa elevar las partes interior y exterior de la ceja (es decir, elevar la ceja por completo). Las tablas de codificación facial más importantes pueden encontrarse en el apéndice C de este proyecto.

FACS también incluye un sistema para describir la intensidad de cada UA mediante la colocación de una letra de A a E después del código numérico, pero antes del signo más. El rango de intensidad para cada letra es: Rastro (A), Leve (B), Marcado / Pronunciado (C), Severo (D), Extremo / Máximo (E). Para continuar con el ejemplo anterior, 1E + 2E representa la elevación extrema (o máxima) de las porciones interior y exterior de las cejas.

Pongamos un caso real que se puede dar en el robot, sea el gesto «sorpresa» el que tiene que expresar el robot Sacarino II. Entonces las unidades de acción[1] que debe de activar serían la 1 intensidad E (Levantar cejar al máximo), levantamiento labio superior 10 a una buena intensidad D, levantar párpado superior 5 a D y por último alzar la cabeza (53) intensidad C. Es decir:

$$Sorpresa \quad = \quad 1E \quad + \quad 10D \quad + \quad 5D \quad + \quad 53C \tag{6.1}$$

En Sacarino II se han implementado las seis emociones básicas además de la expresión neutra, que son: Tristeza, Asco, Miedo, Alegría, Enfado, Sorpresa y Neutra.

Para obtener como resultado cada una de estas expresiones se han definido una serie de AUs (Action Units) a partir del código FACS. Se tiene que aclarar que debido a las limitaciones mecánicas del robot (el fin del robot tampoco es ser un avatar con muchos

[1] Las tablas de códigos faciales más importantes pueden encontrarse en el apéndice C de este proyecto.

músculos faciales), no tiene sentido añadir muchas más UAs puesto que la expresividad es limitada. Sacarino II cuenta con movimiento de párpados, exterior de cejas, boca y cuello. De esta manera se tiene la tabla 6.1.

UNIDADES DE ACCIÓN DEFINIDAS EN SACARINO II

Identificador	**Descripción**
2	Levantamiento exterior de cejas.
3	Bajar cejas.
45	Bajar párpados.
M45	Levantamiento de párpados.
46	Guiño ojo izquierdo.
M46	Guiño ojo derecho.
47	Apertura de ojos.
51	Movimiento cuello izquierda.
52	Movimiento cuello derecha.
53	Alzar cuello.
54	Bajar cuello.

Tabla 6.1: *Unidades de acción en el robot Sacarino II*

6.2. Estructura de programación de los nodos

En el capítulo anterior se explicaba cómo en el sistema de seguimiento se había seguido una programación a distintos niveles, aplicando el concepto de arquitectura. El servicio de bajo nivel ros_ss32 encargado de comunicarse con la placa de servomotores y el conjunto de nodos para el seguimiento, de alto nivel, encargados de la detección y cálculos a partir de los datos recibidos de los sensores Kinect y láser SICK.

En este caso, para el sistema de expresiones se reutiliza el nodo que hace uso del servicio ros_ss32, como el de bajo nivel (0), y, sobre este, se sitúan a su vez otros dos nodos, cada uno en un nivel de abstracción superior al anterior (niveles 1 y 2) 6.1. Se trata de los nodos servidor de unidades de acción (*mua.py*) y del nodo cliente de servidores de acción (*ua_client.py*).

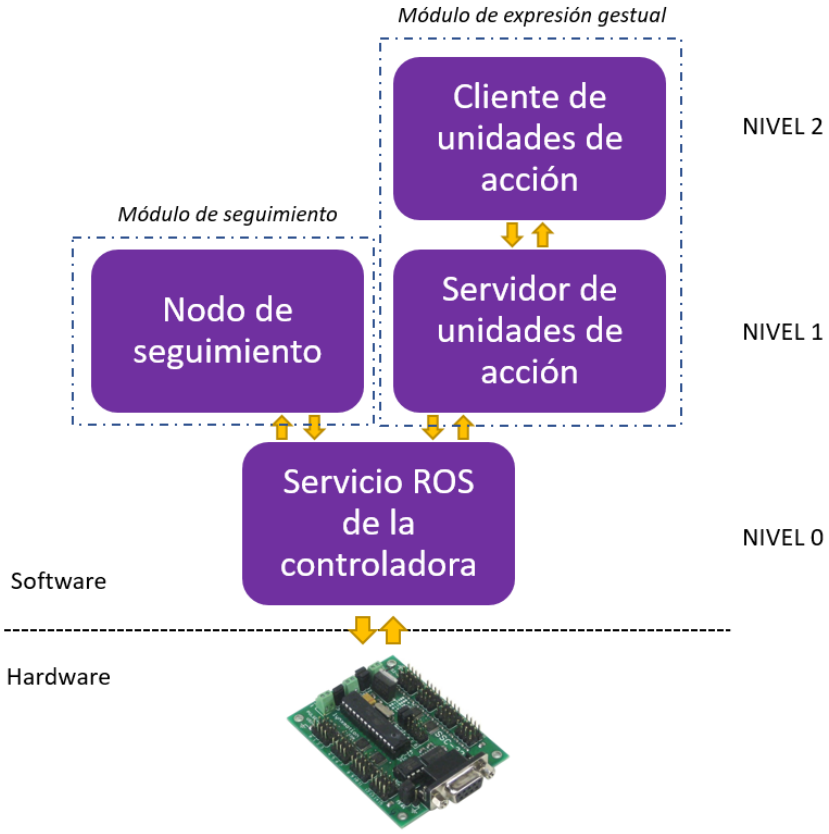


Figura 6.1: Niveles de abstracción en los sistemas

A continuación se describen cada uno de estos nodos, el pseudocódigo que implementan, qué función realizan y la estructura de los mensajes de la que hacen uso.

6.2.1. Nodo servidor de unidades de acción

Como se ha expuesto anteriormente, cada gesto está compuesto por varias unidades de acción (que mueven uno o varios músculos faciales). A su vez, cada unidad de acción lleva asignadas una intensidad y una velocidad. Pues bien, este nodo recibe como entrada un mensaje de tipo vector de unidades de acción (*ua_vec*). La estructura unidad de acción (ua) consta de los siguientes campos descritos en

Tipo de Dato	Nombre del campo	Descripción
string	code	Código de identificador de la unidad de acción.
string	intens	Intensidad de la unidad de acción.
string	vel	Velocidad de la unidad de acción.

Tabla 6.2: *Definición del mensaje ua.*

Una vez se recibe como entrada un vector de tipo ua_vec, el nodo realiza la carga carga un archivo de configuración (este se detalla más adelante en el apartado de interfaces). Este archivo de configuración, contiene información relativa a qué servomotores mueve esa unidad de acción, cuánto ha de moverlos y a qué velocidad. A continuación, se emprende una búsqueda de las unidades de acción, requeridas a la entrada, en la lista de unidades de acción de nuestro archivo. Una vez encontrada ya la UA requerida en nuestro archivo de configuración, el nodo procede a calcular posiciones y velocidades de los servomotores en función de los campos intensidad y velocidad.

Un ejemplo intuitivo de cómo esto último se lleva a cabo es tener en cuenta que el campo intensidad puede tomar 6 valores, 0, A, B, C, D o E, como se ha explicado. Luego si conocemos la posiciones de los servomotores mínima y máxima que representan esa unidad de acción, simplemente se trata de mapear estos límites, esto es tomar el intervalo (máximo - mínimo), dividirlo en seis partes y luego multiplicar una porción tantas veces como indique la intensidad[2].

Este nodo se comunica con el servicio de bajo nivel a partir de las primitivas de la interfaz ya comentadas en 5.4. Además, está suscrito al tópic «ua_client/action_units» y publica en el topic «Succeed» la respuesta recibida por el servidor. El pseudocódigo implementado es el siguiente:

```
Cargar en una lista archivo de configuración .json
 Suscribirse al topic
  Cuando llegue un nuevo mensaje:
    Buscar cada elemento (tipo ua) en la lista
      Si encontrado
      Calcular posición de cada servo
```

[2]El carácter A significa multiplicar por 1, B multiplicar por 2 ,...

```
Calcular velocidad de cada servo
Enviar al servicio
Esperar respuesta del servicio
Publicar respuesta en topic
```

6.2.2. Nodo cliente de unidades de acción

Por encima del nodo servidor de unidades de acción se encuentra el cliente de unidades de acción. Este nodo tiene como principal tarea solicitar al servidor las unidades de acción que definen el gesto que le llega como entrada. Como es de esperar, este nodo recibe como entrada un gesto, a continuación se encarga de buscarlo en un archivo de configuración, esta vez un archivo de configuración de gestos (se ampliará en una sección posterior, cuando se hable de interfaces), y una vez sea encontrado, obtiene como salida el vector de unidades de acción que conforman ese gesto (esto, a su vez, es la entrada del nodo servidor de acciones) junto con sus intensidades y velocidades.

Un objeto tipo gesto (en la programación referido como «gesture»), viene definido por

Tipo de Dato	**Nombre del campo**	**Descripción**
string	name	Código de identificador del gesto.
ua_vec	uas	Vector de unidades de acción.
string[]	intens	Array de intensidades de cada unidad de acción del gesto.
string[]	vels	Array de velocidades de cada unidad de acción asocidada al gesto.
string	mouth	Ruta de acceso al tipo de boca.

Tabla 6.3: *Definición del objeto gesto.*

Quizá el único miembro que puede resultar desconocido es el campo «mouth» (boca). Como se introducía en el segundo capítulo, Sacarino II tiene por boca un display en matriz de ledes, que se iluminan de unas formas u otras para emular diferentes muecas. Dicho display funciona con un nodo ROS llamado «led_matrix», al que se le manda simplemente la ruta de acceso a un directorio donde se encuentran diversos archivos

.msg que no son más que matrices de datos expresadas en hexadecimal indicando los leds que deben encenderse.

Ua_client está suscrito a un topic del servidor de «actions»(tratado en el siguiente capítulo) llamado «Working_mode» y publica en el topic «ua_client/action_units». El pseudocódigo que ejecuta se muestra a continuación:

```
Cargar en una lista archivo de configuración .json
  Subscribirse al topic
   Cuando llegue un nuevo mensaje (gesto):
     Buscar por nombre el gesto en la lista
       Si encontrado
       Empaquetar la cadena de unidades de acción en un vector
       Empaquetar la cadena de intensidades de cada ua en un vector
       Empaquetar la cadena de velocidades de cada ua en un vector
       Publicar en el topic los vectores
```

6.3. Herramientas para la configuración

Sería poco riguroso, además de restar legibilidad, si se tuviera que introducir en el propio código de los nodos todos los números referentes a límites de velocidad y posición, códigos de unidades de acción, la correspondencia entre unidades de acción y servomotores, etc... Es por ello, que cada nodo carga un archivo de configuración cuando es ejecutado por primera vez, donde está reflejada toda esta información.

La información en estos ficheros de configuración está guardada en el formato que se conoce como estándar JSON.

6.3.1. Estándar JSON

JSON, acrónimo de *Java Script Object Notation*, es un formato sencillo de texto para el intercambio de datos. Los tipos de datos disponibles son los siguientes:

- Números: pueden ser números tanto positivos como negativos aparte de poder tener parte decimal siempre separada por puntos. Ejemplo: 457,589.
- Cadenas de caracteres: representan secuencias de ningún o varios caracteres. Se escriben entrecomillados además de permitirse cadenas de escape. Ejemplo: "Hola".
- Variables booleanas: representan valores booleanos y pueden tener dos valores , 1 true o 0 false, en lógica positiva.
- Null: representa vacío o un valor nulo.
- Array: es una lista ordenada de cero o más elementos los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes. Ejemplo ["Juan", "Mario,","Marta",..].
- Objetos: son colecciones desordenadas de pares tipo <nombre>:<valor>puestas entre llaves y separados por comas. El nombre tiene que ser una cadena y entre ellas. El campo valor puede ser de cualquier tipo.

Un ejemplo de archivo formato JSON es el mostrado en la figura 6.2.

Figura 6.2: Ejemplo texto en formato JSON

En el proyecto se ha usado este tipo de representación para guardar distintas configuraciones. En primer lugar, tenemos un archivo de configuración para el nodo to_servo donde se detallan por una parte los parámetros referentes a la conexión con la contro-

ladora: la frecuencia de transmisión por USB a la placa en baudios, la ruta al puerto serie donde se ha conectado la placa al PC y el retardo de sondeo. Por otra parte, se declara en un vector de objetos las constantes referentes a los límites de los servomotores conectados, su número de identificación (id), la velocidad máxima y el ángulo inicial. Véase la figura 6.3.

De manera análoga, se tiene otro archivo de configuración similar que asigna a cada unidad de acción un conjunto de servomotores (cargado en el nodo ua_server), junto con sus posiciones máximas y mínimas de ese servo en esa unidad de acción. Esto último es importante, no se debe confundir el rango total de movimiento del servomotor (p. ej de -90 º a 90º) que vendría limitado por las especificaciones del fabricante o por límite mecánico del robot, con el rango que tiene un determinado servomotor en una unidad de acción. De esta manera, si para expresar alegría se requiere la unidad de acción «subir párpados», el rango del servomotor para esta unidad de acción podría ser de 0º (suponemos que 0º es el párpado medio cerrado) a 90 º (suponiendo 90º como apertura completa).

```
"ssc32_cfg": {
  "dev_name": "/dev/ttyUSB0",
  "dev_baud": 9600,
  "poll_delay": 0.01
},
"servo_cfg": [
  {
    "title": "Cuello horizontal",
    "vel_max": "0.5",
    "init_ang": -0.51,
    "min_ang": -1.5,
    "max_ang": 1.57,
    "id": "16"
  },
  {
    "title": "Cuello vertical 1",
    "vel_max": "0.5",
    "init_ang": 0.0,
    "min_ang": -1.48,
    "max_ang": 1.48,
    "id": "17"
  },
```

Figura 6.3: Archivo de configuración *servos.json*

En este caso, para cada unidad de acción nos interesa saber el conjunto de identi-

ficadores de los servomotores, y, para cada servomotor, su posición inicial, mínima y máxima. El archivo tiene por nombre «actionunits.json» y se muestra en la figura 6.4a.

Por último y para finalizar este apartado, se ha de comentar el archivo cargado en el nodo cliente de unidades de acción. En este caso se trata de almacenar información relativa a los gestos tales como los parámetros nombre, unidades de acción que lleva asignados, intensidades de cada unidad de acción y velocidades. El archivo se llama «gestures.json» y su estructura se ilustra en la figura 6.4b.

(a) Archivo de configuración *actionunits.json*

(b) Archivo de configuración *gestures.json*

Figura 6.4: Archivos de configuración JSON

Capítulo 7

SISTEMA GLOBAL DE INTERACCIÓN E INTERFAZ GRÁFICA

El presente capítulo expone el sistema global de interacción que gobierna ambos módulos, seguimiento y expresión gestual, siendo el nivel más alto en la arquitectura. Por otro lado, se presenta la interfaz gráfica desarrolla como herramienta de ayuda a la configuración del subsistema de expresión. Es, por tanto, un capítulo donde se integran los conceptos desarrollados en capítulos anteriores.

7.1. Introducción al sistema global de interacción

El control de los subsistemas desarrollados anteriormente puede ser complejo si se intentan gobernar por separado. Con esta justificación, se propone la inclusión de una última capa en la arquitectura que ejerza de interfaz entre el secuenciador del robot y el sistema de interacción, de forma que todo lo que pase y cómo se haga permanezca inadvertido al exterior, es decir, se trata de facilitar la comunicación secuenciador - sistemas de seguimiento y expresiones. Como consecuencia, se han definido varios «Modos de funcionamiento»:

- Modo interacción: el robot realizará seguimiento en el plano panorámico y efectuará las expresiones que sean ordenadas por el secuenciador. En este caso, en el seguimiento se hace uso del láser SICK.
- Modo seguimiento (*Tracking*): mediante el sensor Kinect, el robot centrará el hacer seguimiento en los planos panorámico y de cabeceo. Durante este modo de funcionamiento el sistema de expresión gestual mantendrá el gesto en la expresión neutra.
- Modo descanso: En el que el robot se lleva a una posición de equilibrio donde ambos subsistemas se mantienen en reposo.

7.1.1. Jerarquía del sistema global

Esta última capa de la arquitectura se sitúa en el nivel más alto, el nivel 3, como se muestra en la figura 7.1. Por encima queda situado el secuenciador de programa del robot, que, de acuerdo a los estímulos externos que considere, irá alternando entre los diferentes modos de funcionamiento definidos en el apartado anterior.

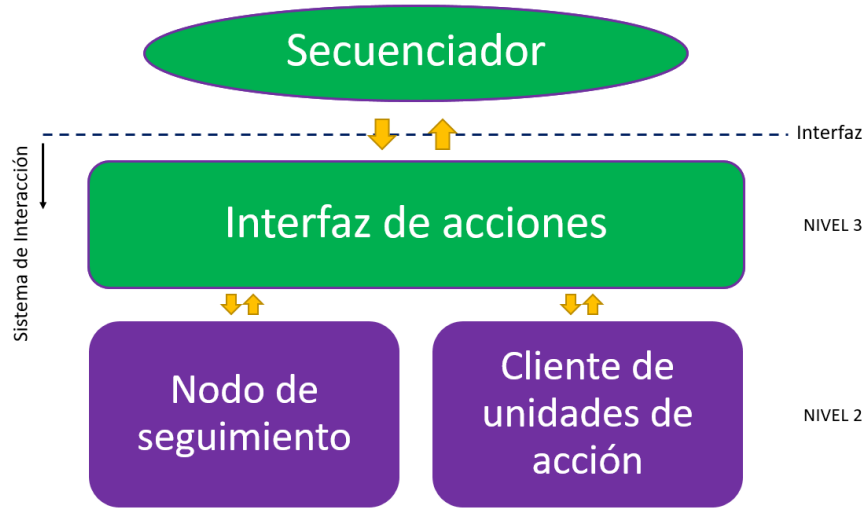


Figura 7.1: Última capa en la arquitectura

ROS proporciona una herramienta muy potente en la que se basa esta última capa, se describe a continuación.

7.1.2. Biblioteca Actionlib de ROS

Como ya se ha descrito anteriormente, en los servicios de ROS, el usuario implementa un modelo de interacción basado en petición/respuesta ente dos nodos, pero no se tienen en cuenta otros aspectos como el caso en que las respuestas tarden un tiempo excesivo en producirse, o que, si el servidor no ha terminado aún con su trabajo, entonces el cliente ha de esperar bloqueado hasta que finalice.

Existe, por tanto, otro método dentro de ROS en el cual se puede cancelar una petición hecha por un cliente y lanzar otra si, por ejemplo, la respuesta no ha llegado en el tiempo que se esperaba. La biblioteca Actionlib [24] provee una forma estándar de implementar este tipo de matices. Actionlib es altamente usada en la navegación de ambos, brazos robóticos y robots móviles.

Como en los servicios de ROS, en actionlib se debe de especificar la acción que se quiere realizar. Dicha especificación es declarada en fichero con extensión .action (de manera similar a la creación de mensajes). El fichero de acción contiene la siguiente información:

- **Goal**: El cliente de acciones (en nuestro caso, el secuenciador) puede enviar un objetivo (goal) que tiene que ser ejecutado por el servidor de acciones. Esto es similar a la petición en un servicio. Por ejemplo, si una articulación de un brazo robot tiene que moverse de 45º a 90º, el objetivo aquí sería 90º.

- **Feedback**: Cuando un cliente de acciones envía un objetivo al servidor, este comenzará a ejecutar su rutina de acuerdo a su función de *callback*. Además de esto, el servidor tiene la opción de informar al cliente del estado de su objetivo. Siguiendo con el ejemplo anterior, un posible feedback puede puede ser el aviso de la llegada a ciertas posiciones intermedias.

- **Result**: Después de completar un objetivo, el servidor de acciones publica un resultado final, puede ser un mensaje para su cómputo, o, simplemente un mensaje de reconocimiento. En el ejemplo, puede ser cualquier mensaje que advierta de que se ha llegado a la posición final.

7.1.3. Implementación del servidor de acciones

El sistema global de interacción se basa en el modelo actionlib descrito en el apartado anterior. En nuestro caso, el secuenciador se trataría del cliente de acciones y, el servidor de acciones sería el encargado de gestionar la comunicación con el resto del sistema para satisfacer los objetivos que mande el secuenciador.

El mensaje de acción que se ha definido contiene los campos mostrados en la Tabla

El objetivo queda definido por el par de campos «mode» y «gesture» de acuerdo a la necesidad del problema, donde una orden del cliente tiene que especificar un modo de funcionamiento y un gesto (aunque este último campo solamente sea utilizado cuando se trate del modo interacción).

Como realimentación al cliente se tiene el campo «percent complete » o porcentaje completado. Se trata de una cadena de caracteres en la que se lleva la cuenta del estado del procesamiento del objetivo. Este porcentaje se corresponde con un temporizador que se ha calibrado experimentalmente midiendo los tiempos que tarda el robot en ejecutar cada gesto.

Finalmente como resultado, el servidor publica un «Éxito» o «Fallido» en función de si se ha llegado a completar el objetivo o no. Esto no solamente tiene que ver con la finalización del temporizador, puesto que la información sería poco fiable, sino que el servidor queda a la espera de recibir un mensaje por parte del nodo de bajo nivel (servidor de unidades de acción) de éxito. La forma de gestionar el éxito o no del nodo de bajo nivel, se basa en que, cuando se solicita al servicio de la controladora de servomotores un movimiento, esa función (move_angle) devuelve una variable booleana, por lo que, si para el movimiento de algún servomotor, devolviera «False» entonces publicaría en el topic «Fallido», dándose así por enterado el servidor de acciones de que la instrucción no se ha llegado a completar y por tanto, publicar el correspondiente mensaje de error en «result».

7.2. Interfaz gráfica. Concepto y justificación

Algo de lo que no se ha hablado aún es de la necesidad de gestionar todos los archivos de configuración descritos en el capítulo 6 (servos.son, actionsunits.json y gestures.json) de manera eficiente.

La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface), es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador[1].

Se trata por tanto de desarrollar un entorno gráfico intuitivo que permita al usuario la rápida modificación y testeo de todos los parámetros contenidos en los ya mencionados, archivos para la configuración. Desde el primer momento en que se ha comenzado a desarrollar la interfaz gráfica, se han tenido claros los siguientes puntos:

- Tenía que ser intuitiva y de acceso rápido a los distintos menús.
- Se debía desarrollar lo más generalista posible. Esto significa que se debe poder transportar a la configuración de otros robots de manera cómoda.

7.2.1. Herramienta WxWidgets

WxWidgets es un entorno de trabajo de código abierto en C ++ que permite escribir aplicaciones gráficas multiplataforma con aspecto nativo en C ++ y otros lenguajes.

WxWidgets fue desarrollado originalmente por Julian Smart en el Instituto de Aplicaciones de Inteligencia Artificial de la Universidad de Edimburgo, para uso interno, y se puso por primera vez a disposición del público en 1992, con una versión muy mejorada lanzada en 1999. La última versión principal de la librería es la 3 y fue lanzada en 2013. En la actualidad wxWidgets se desarrolla y mantiene por Julian Smart, Vadim Zeitlin, Stefan Csomor, Robert Roebling, Vaclav Slavik y muchos otros.

[1] https://es.wikipedia.org/

En comparación con las otras librerías similares, wxWidgets:

1. Es la única librería de C ++ GUI construida para trabajar con widgets GUI en su forma nativa, lo que se traduce en la mejor experiencia de usuario en cada plataforma.

2. Está escrita usando sólo el estándar de C ++ y no se basa en ninguna extensión o procesamiento previo encargo.

3. Es de código abierto y libre para el uso en proyectos de código abierto y comerciales.

WxWidgets proporciona una API(Application Programming Interface) intuitiva. También es robusto y estable, y las aplicaciones escritas usando wxWidgets 2.0 de hace casi 20 años todavía se puede construir hoy con wxWidgets 3 casi sin cambios. wxWidgets tiene una comunidad grande y activa, incluyendo tanto los usuarios como los desarrolladores de la librería. También está disponible ahora para más de una docena de otros idiomas, como Python, Perl, Ruby, Lua, Haskell, D, Erlang, PHP, además de C ++.

A continuación se presentan varias ventajas del uso de WxWidgets: disponible en todas las plataformas de escritorio principales, libre para cualquier uso, sistema de eventos flexible, impresión, ayuda, portapapeles, arrastrar y soltar, documento/vista, reproductor potente de edición de texto con resaltado de sintaxis [25].

7.2.2. WxPython

La interfaz desarrollada ha sido programada nodirectamente con WxWidgets sino sobre WxPython. WxPython es un conjunto de herramientas GUI para el lenguaje de programación Python. Se permite, de esta forma, a los programadores de Python crear programas con una interfaz gráfica de usuario robusta, altamente funcional, simple y fácil. Se implementa como un módulo de extensión de Python (código nativo) que abarca la popular librería wxWidgets multiplataforma, que está escrita en C ++.

Como Python y wxWidgets, wxPython es de código abierto lo que significa que es libre utilización y el código fuente está disponible para que cualquiera pueda mirar y

modificar. O para que cualquier persona pueda contribuir con soluciones o mejoras al proyecto [26].

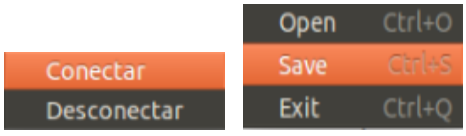
7.2.3. Instalación en Ubuntu 14.04 LTS

La distribución de Linux con la que está trabajando en este proyecto es Ubuntu en su versión 14.04 LTS. Para la instalación de WxPython, bastan con introducir por consola de comandos[2]:

apt-get install python-wxgtk2.8

7.2.4. Interfaz gráfica desarrollada

Lo primero que nos encontramos en la parte superior es la barra de menús desplegables, que contiene diversas opciones como ayuda, conexión/desconexión al servicio de servomotores, abrir archivos, guardar y salir de la interfaz.



(a) Desplegable de la pestaña «servicio ROS»

(b) Desplegable de la pestaña «opciones»

Figura 7.2: Distintos desplegables del menú superior

El resto de la pantalla de la interfaz gráfica está dividia en tres paneles bien diferenciados, separados por separadores (*splitters*). Empezando desde arriba hacia abajo, la primera división horizontal que nos encontramos tiene que ver con toda la información relativa a la configuración de parámetros de los servomotores, de bajo nivel. Este panel, a su vez queda dividido en dos secciones, una izquierda y otra derecha. La sección izquierda consta de tres botones: «Guardar JSOn», «Cargar JSON» y «Añadir a AU». El primero de los botones sirve para salvar la configuración actual que estemos

[2]Para compilar desde código fuente consultar [26]

haciendo de los servomotores. El botón de cargar, lo que hace es volcar en el programa la información de un archivo de configuración con extensión .json de servomotores (en nuestro caso, el archivo servos.json). El botón de «añadir a Au» añade un servomotor seleccionado a la unidad de acción seleccionada en ese momento, que se encuentra en el panel inmediatamente inferior.

Por otro lado, en la sección derecha se encuentran, en primer lugar unas cajas cuyo valor es editable y que cada una definen un parámetro del objeto servomotor. Estos parámetros son: Nombre, vel_max, init_ang, min_ang, max_ang y el número de identificador en la controladora. Cada uno de estos parámetros, como se ha dicho, lo introduce manualmente el usuario. Una vez ha introducido en todos los campos al menos un valor (el valor por defecto es 0 sino se rellena), entonces para guardar los cambios debe pulsar el botón inferior de «Actualizar valores», y, automáticamente, los valores del servomotor cambian a los seleccionados en ese momento.

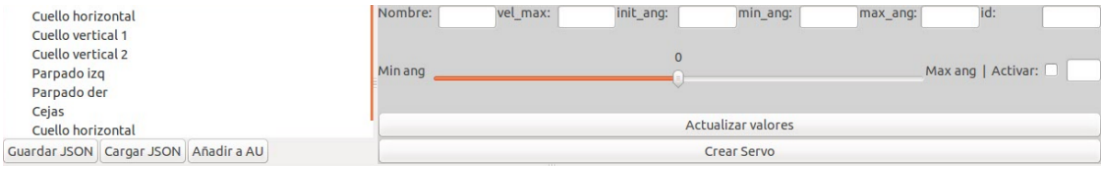


Figura 7.3: Panel de configuración de servomotores

Para verificar si una configuración es adecuada o no, el usuario puede comprobar el movimiento de los servomotores. Para ello, se ha de conectar primeramente al servicio de servomotores (que se supone previamente arrancado) pulsando el botón destinado al efecto en la barra superior /ROS/Conectar al servicio. A continuación, se debe hacer click en el box situado a la derecha del *slider* (barra deslizante). Finalmente, basta con mover la barra deslizante hacia un lado y a otro para ir modificando la posición del servomotor, de este modo, se pueden ir ajustando los parámetros para la verificación de límites mecánicos, velocidades ...

El siguiente panel horizontal que nos encontramos (ocupa la parte central) es donde se define toda la configuración para las unidades de acción. De manera análoga al panel de servomotores, se tiene a su vez dos secciones, izquierda y derecha. En la sección izquierda es donde va situado el árbol de unidades de acción que el usuario va

añadiendo y configurando. Contiene tres botones, «Guardar aus JSOn», «Cargar aus JSON» y «Añadir a Gesto». Los primeros dos botones cargan y guardan un archivo de configuración .json, el archivo en el proyecto está nombrado como actionunits.json. Con el último de los botones se tiene la posibilidad de añadir la unidad de acción seleccionada en la sección izquierda de este panel al gesto seleccionado en el panel inmediatamente inferior.

Figura 7.4: Panel de configuración de unidades de acción

En la sección derecha del panel de unidades de acción nos encontramos con varias etiquetas editables: init_pos, min_pos, max_pos y servo. Donde las tres primeras se refieren a límites y la última al servo que se está configurando. Esto es importante, como ya se explicaba en el capítulo 6, cada unidad de acción mueve uno o varios servomotores, pero los límites de un servomotor dentro de una unidad de acción no tienen por qué coincidir con los límites absolutos del servomotor, de hecho, lo normal es que no sean los mismos.

Como cabe esperar se tiene un slider (barra deslizante) y un grupo de etiquetas modificables para cada servomotor dentro de una unidad de acción. Además, en la parte inferior de la sección derecha se muestra una botonera cuyo nombre es autoexplicativo.

Por último, en el tercer panel (espacio inferior de la pantalla) se trata toda la configuración referente a los gestos. Análogamente a los paneles anteriores se dispone de dos secciones. La sección izquierda es donde se muestra el árbol de todos los gestos con los que se está trabajando. Se pueden salvar y cargar de un fichero utilizando los botones destinados a tal efecto en su parte inferior.

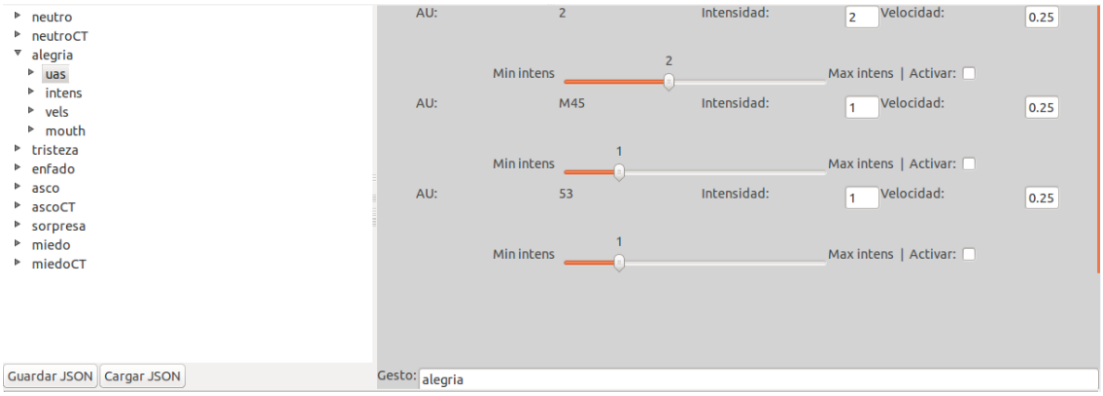


Figura 7.5: Panel de configuración de gestos

En la sección derecha del panel, se disponen las etiquetas de configuración para un gesto, las cuáles son «AU»,Intensidad,Velocidad. «AU» quiere decir (action unit), cada gesto está comprendido por una o varias unidades de acción. De acuerdo al FACS, a cada unidad de acción le está asignada una intensidad de la «A» (rastro) a «E» (severo) , por lo que se requiere del campo «Intensidad». Por último, se guarda un campo de velocidad que puede tomar tres valores «L» lento, «M» medio y «R» rápido, que es con la velocidad que se moverán los servomotores de cada unidad de acción.

Para cada conjunto de configuración de unidad de acción se tiene un slider (barra deslizante) que, esta vez no se trata de ángulos, sino de indicar del 1 al 5 nivel de intensidad. Además, para terminar de definir un gesto, como Sacarino II dispone de una pantalla de ledes como boca, se ha incluido un apartado para guardar el tipo de boca que se mostrará en la pantalla cuando se ejecute ese gesto, basta con seleccionarla haciendo click en la que se desea. Finalmente, se tiene una botonera análoga al panel de unidades de acción.

Capítulo 8

RESULTADOS Y VALIDACIÓN

En este capítulo se analizan los resultados obtenidos con las estrategias finales implementadas en el sistema de interacción desarrollado, para comprobar el grado de consecución de los objetivos marcados.

8.1. Subsistema de seguimiento

Según el modo de funcionamiento en que se encuentre el robot, se utiliza la combinación de los sensores Kinect y láser SICK para realizar el seguimiento.

8.1.1. Seguimiento con el láser SICK

El seguimiento con el láser SICK, ha demostrado ser preciso en el posicionamiento en plano panorámico, incluso en el caso de que varias personas se encuentren frente al robot. Además, se debe saber que el entorno en el que se han diseñado los experimentos se trata de un espacio reducido y donde existen muchos elementos que podrían dar lugar a que el láser confundiera personas con otros objetos o ruido.

Para el experimento con el láser se han dispuesto el número de personas que cabe esperar frente al robot en un escenario real de interacción similar al planteado. El robot, como era de esperar, en un principio seguía a un usuario de manera exclusiva.

Seguidamente, la persona seguida salía del rango de alcance del láser, obligando a que éste cambiara el seguimiento a otra de las dos personas (se escoge la más cercana). Finalmente, tras el paso de un corto período de tiempo, la persona seguida quedaba estática y la restante comenzaba a experimentar cambios significativos de posición. En este momento, el robot se centraba en este último usuario.

El umbral de 0.25 (25 %) en la fiabilidad, ha probado ser suficiente para el rechazo de seguimiento a ruido en prácticamente todas las situaciones (mayor al 95 %), pues el resultado de la detección, a menudo, estaba contaminado por ruido o detección de falsos positivos. De este problema se tenía que hacer cargo la etapa de seguimiento, discriminando si seguir o no en función del umbral de fiabilidad. Por otro lado, si, por lo que fuese, el robot seguía un objeto no humano, éste corregía el seguimiento tan pronto como una persona en frente de él experimentara un cambio de posición significativo.

8.1.2. Seguimiento con Kinect

En este caso, Kinect realiza seguimiento en ambos planos: panorámico y cabeceo. La descripción del entorno donde se han desarrollado los experimentos merece la misma descripción que el entorno del láser SICK tratándose de un espacio lleno de detalles como armarios, baldas, cajas, ordenadores, paneles de herramientas, etc, fácilmente asimilables como falsos positivos.

El experimento que se ha llevado a cabo en este caso es la detección y el seguimiento de un usuario, realizando éste tanto cambios bruscos de posición (señales escalón) como cambios progresivos (señales rampa) en ambos planos (panorámico y cabeceo).

A lo largo del capítulo 5 se propusieron varias estrategias para hacer seguimiento con Kinect. La estrategia basada en modulación de velocidad se ha mostrado inestable. Puesto que, si se demandaban movimientos a altas velocidades de la cabeza, Kinect no tenía tiempo de muestrear antes de que el usuario hubiese salido del campo visual, y, como resultado, la cabeza no se detenía en ningún momento acabando siempre en las posiciones extremas de los servomotores (se recuerda que la referencia de posición que se enviaba siempre en esta estrategia eran la máxima o mínima para ese servomotor,

en función del signo de los datos que se extraían con Kinect).

Por otro lado, las estrategias por filtrado de referencia de posición han demostrado ser más fiables. En el caso del filtrado lineal del error se han obtenido buenos resultados, pero, puesto que los cambios de posición del usuario son imprevisibles, los esfuerzos de control cuando estos cambios eran bruscos y cortos en el tiempo, se producía un sobrepaso de la posición final, antes de volver a ésta. Esto es un elemento que tiene un impacto negativo sobre la interacción, deteriorándola.

La estrategia que se ha escogido finalmente ha sido la de *filtrado de la referencia de posición con tratamiento no lineal del error*. La función no lineal tipo sigmoide tiene un doble efecto sobre el sistema:

1. Convierte la respuesta del sistema en sobreamortiguada, evitando sobrepasos en el posicionamiento, al necesitar de cambios significativos en la señal de error para comenzar a modificar la señal de control.

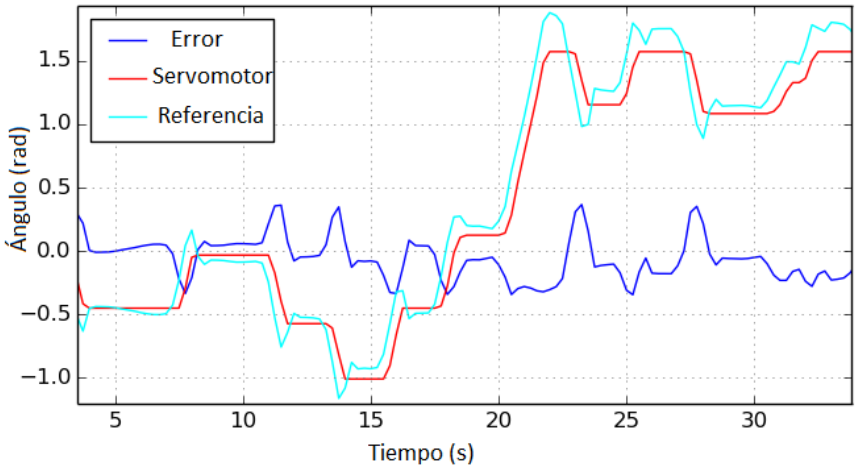
2. Previene al sistema de estarse ajustando continuamente ante pequeños movimientos del usuario. Además de tener un buen impacto en la interacción, esto tiene consecuencias positivas sobre la vida de los actuadores (servomotores).

Para proceder al análisis cuantitativo del seguimiento con dicha estrategia, un buen indicador es la representación de las gráficas de error, movimientos del usuario, y respuesta del sistema a lo largo del tiempo, en ambos planos, panorámico y cabeceo.

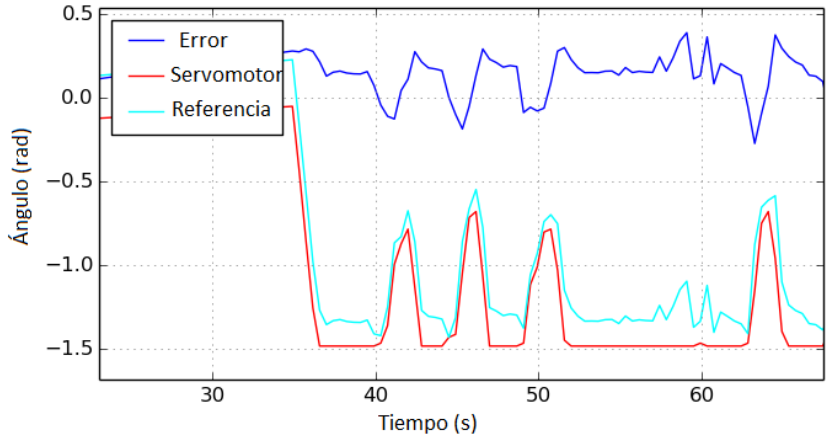
Como se observa en la figura 8.1, la gráfica representada en azul oscuro es la gráfica de error, en la que se confirma que éste tiene que ser significativo para que la señal de control modifique la posición. Además, el sistema de control es capaz de contener el error en un banda de $\pm$ 0.1 rad en el estacionario, suficiente para que, en la realidad, el robot quede perfectamente orientado al usuario.

La gráfica coloreada de azul claro es la correspondiente a los movimientos del usuario. Se demuestra que los cambios que experimenta son rápidos e imprevisibles. Por último, la gráfica coloreada en tojo corresponde a la respuesta del sistema. Se observa que el máximo retardo en el seguimiento es de 0.15 segundos, más que aceptable teniendo en cuenta el peso de la cabeza y las características mecánicas. Por otro lado, se confirma que

la respuesta es sobreamortiguada, teniendo un impacto muy positivo en la interacción.



(a) Seguimiento en plano panorámico



(b) Seguimiento en plano cabeceo

Figura 8.1: Respuesta temporal del seguimiento con Kinect

8.1.3. Validación

Se recuerda que uno de los objetivos generales era el de «lograr un movimiento natural de la cabeza del robot tal que sea capaz de mantener la mirada fija en el usuario, es decir, todo aquello que tiene que ver con la orientación y el seguimiento de la cabeza del robot.».

Con la estrategia elegida, el diseño de experimentos realizado y el análisis de resulta-

dos, Sacarino II es capaz de hacer seguimiento a la persona con la que está interactuando de manera fluida y lo más cercana posible a cómo interactuarían dos humanos. Por lo tanto tanto, podemos dar este objetivo como validado y, por ende, el subsistema.

8.2. Subsistema de expresiones

El subsistema desarrollado consigue expresar seis emociones básicas además de la expresión neutra, que son: miedo, tristeza, asco, alegría, miedo y sorpresa.

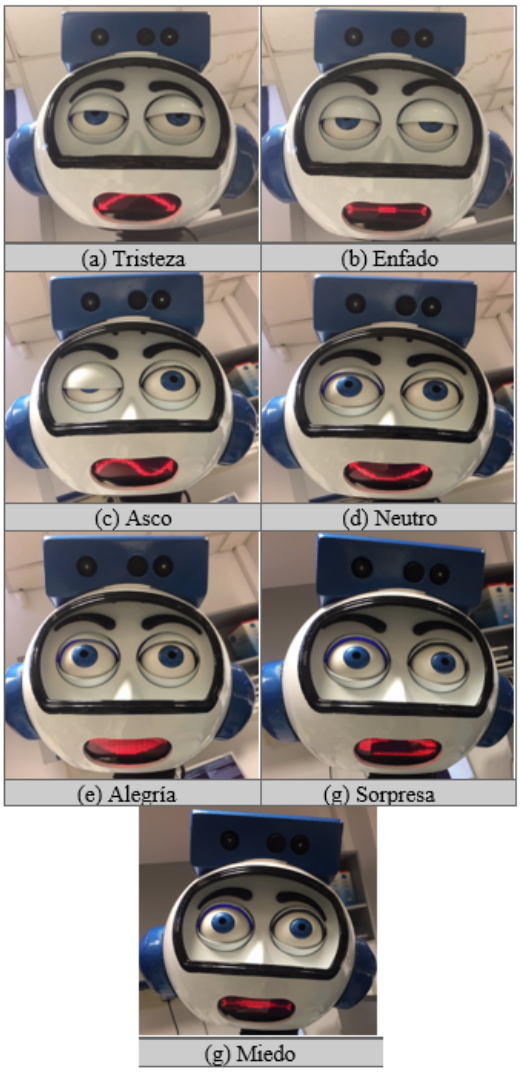


Figura 8.2: Emociones en Sacarino II

Para ello se ha hecho uso del estándar FACS, aprovechando al máximo la expresividad de la que dispone el robot.

Como se puede ver en la figura 8.2, para la emoción de tristeza se necesitan bajar cejas a una intensidad alta, ligeramente cerrar los párpados, una mueca característica en la boca y bajar el cuello.

En el caso del enfado, predomina un levantamiento de cejas marcado, achinar un tanto los ojos y una boca seca y plana además de bajar el cuello. Por su parte, en la emoción de asco destacan un levantamiento de cejas moderado, guiño de un ojo mientras se mantiene bien abierto el párpado del otro, una mueca de disgusto y un giro de cuello leve hacia arriba y a un lado.

Si lo que se pretende expresar es alegría, entonces se deben levantar levemente las cejas, abrir los párpados una intensidad elevada, levantar ligeramente el

cuello y poner una mueca sonriente, eso es lo que se muestra en 8.2 (e). Por otro lado, en la emoción de sorpresa intervienen la unidad de acción de levantamiento de cejas a intensidad máxima, un levantamiento de párpados a intensidad máxima, así como una elevación marcada de cuello a alta velocidad y boca abierta.

Finalmente, para expresar miedo es necesario bajar las cejas una intensidad moderada, abrir párpados en una intensidad máxima, levantar levemente el cuello y poner los labios en tensión.

8.2.1. Validación

El segundo de los objetivos generales para este proyecto era: «Una vez iniciada una interacción con un usuario, el robot debe de poder expresar gestos y emociones.». La experiencia cuando se trata con el robot es agradable, lográndose reconocer cada una de las emociones implementadas, por lo que se da el objetivo como cumplido, así como al subsistema. Además, uno de los objetivos secundarios era el de adaptarse a los estándares para un desempeño generalista. Esto se ha cumplido en el uso del estándar de codificación facial FACS.

8.3. Interfaz gráfica (GUI)

Para el diseño se la interfaz gráfica se ha utilizado WxPython, dando como resultado una herramienta de uso intuitivo y sencillo para el usuario. Como se observa en la figura 8.3, la interfaz está dividida en tres paneles horizontales y, a su vez, en sendas secciones izquierda y derecha. El primer panel está dedicado a la configuración, mientras que el panel central se centra en el tratamiento de las unidades de acción y, finalmente, en el último panel se configura todo lo relacionado con los gestos y emociones.

Por otro lado, en las secciones de la izquierda se muestran, en forma de árbol desplegable, los diferentes ítems que se van añadiendo, ya sean servomotores, unidades de acción o gestos. Dentro de las secciones de la derecha se pueden configurar una serie de etiquetas (*tags*) dependiendo de tipo de ítem tratado, por ejemplo, posiciones iniciales, velocidades, intensidades, etc. Para finalizar se incorporan herramientas de

carga/guardado de ficheros así como control de fallos en forma de ventanas emergentes o cuadros de diálogo.

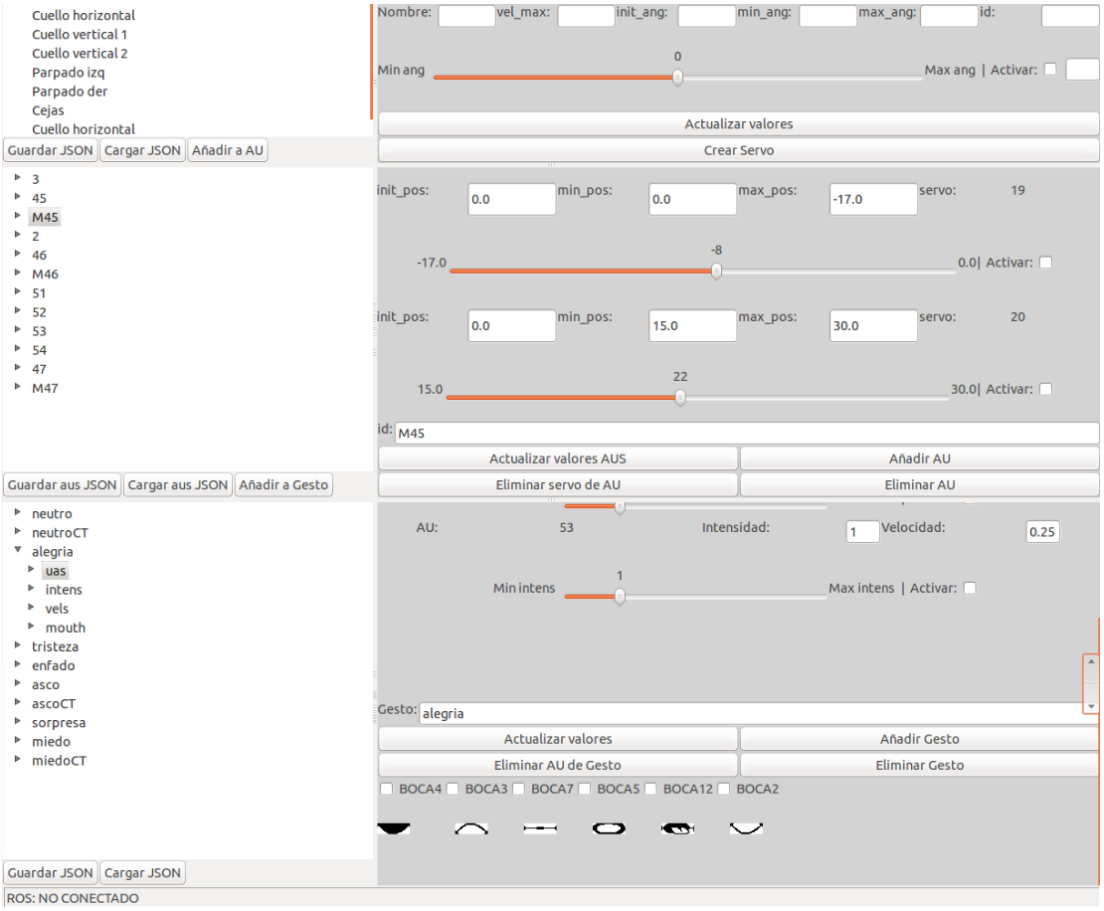


Figura 8.3: Interfaz gráfica

8.3.1. Validación

Finalmente, el último objetivo marcado era el de desarrollar una interfaz gráfica (GUI) intuitiva para el usuario. Analizando la interfaz gráfica final se desprende el uso sencillo y aprendizaje rápido, por lo que se puede dar por válido el cumplimiento del objetivo. Por otra parte, al tratarse de una interfaz generalista, es fácilmente extrapolable a otros robots, dándose así por satisfecho uno de los objetivos secundarios.

Capítulo 9

ANÁLISIS ECONÓMICO DEL PROYECTO

El proyecto abarca la implementación del sistema del sistema de visión artificial sobre Sacarino II, es decir, el estudio económico que se expone a continuación cubre exclusivamente el alcance de este sistema, no del robot completo. Aclarado esto, en este capítulo se detalla una aproximación al coste de la realización del proyecto para determinar cómo es de viable y si en verdad resulta ventajoso económicamente frente a otros trabajos desarrollados en el mismo campo. Para ello, hay que tener en cuenta tanto costes directos como material como indirectos, costes de personal, mantenimiento, consumos eléctricos y desplazamientos.

En la realización de este apartado han sido tenidos en cuenta otros estudios en este campo [9], [2].

9.1. Recursos empleados

Como en cualquier proyecto de ingeniería existen una serie de recursos tanto físicos (hardware) como programas informáticos (software). Incluimos en este apartado ordenadores, sensores, programas utilizados, sistemas operativos y material de oficina. Bien es cierto que gran parte de este proyecto hace uso del llamado software Open Source,

software libre y abierto pero que a la vez cubre de sobra las necesidades y funcionalidades que se quieren implementar, lo que reduce de manera considerable el costo del sistema.

En cualquier caso, el proyecto ha abarcado en su totalidad las siguientes herramientas, independientemente de si supondrán un coste computable o no:

- Equipamiento físico: ordenador sobremesa HP Compaq.
- Sistemas Operativos empleados: WINDOWS VISTA PROFESSIONAL, Ubuntu 14.04 y ROS(Robot Operating System) Indigo. Aquí se incluye también el uso de librerías ssc32 controller, openni, Nite.
- Sensores empleados: Microsoft Kinect, se trata de un sensor bastante robusto a un precio asequible. A ello hay que añadir que existe una extensa comunidad que desarrolla software libre haciendo uso de Kinect, lo que abarata aún más su coste. Otro sensor empleado ha sido el láser LMS100 SICK para visión 2D auxiliando a Kinect en la operación de seguimiento de usuarios.
- Editor de texto Office Writer.
- Actuadores empleados: seis servomotores de los cuáles tres tienen engranaje metálico. Tres del modelo MG90s (que se encargan del movimiento de párpados y cejas) y otros tres servos de alto torque para ganar los tres grados de libertad al cuello.

9.2. Costes directos

Aquí se encuentran aquellos costes imputables que están ligados de manera directa al desarrollo del presente proyecto, como puede ser la pano de obra utilizada, el material directamente empleado en el sistema desarrollado o la amortización de equipos y programas en su uso en el trabajo como parte de su vida útil.

9.2.1. Costes de personal

La realización del proyecto ha sido llevada a cabo por un ingeniero al que le han sido asignadas las tareas de búsqueda de información previa, diseño e implementación del sistema de visión artificial y expresión gestual.

COSTE ANUAL

Sueldo Bruto	23.015,69	€
Seguridad social (35 % sueldo)	8055.49	€
Total	31071,18	€

Tabla 9.1: *Salario anual*

Otro ingeniero ha prestado ayuda en determinados aspectos de desarrollo, por lo que también tiene que ser tenido en cuenta, a pesar de ser ayuda externa. Para hacer el cómputo de estos costes, hay que tener en cuenta el salario, bruto[1] y anual así como la cotización a la seguridad social (35 % del sueldo bruto) y los días que representa este salario[9] (tablas 9.1 y 9.2).

DÍAS EFECTIVOS POR AÑO

Año Promedio	365,25	días
Sábados y Domingos	-104,36	días
Días de vacaciones	-20,00	días
Días festivos reconocidos	-15,00	días
Días perdidos estimados	-5,00	días
Total días efectivos estimados	229.89	días

Tabla 9.2: *Días Efectivos por año*

Una jornada laboral es de 8 horas, por lo que el cálculo total de horas de trabajo es el siguiente:

$$220,89\frac{\text{días}}{\text{año}} \cdot 8\frac{\text{horas}}{\text{día}} = 1{,}767,12\frac{\text{horas}}{\text{año}} \tag{9.1}$$

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

$$\frac{31{,}071,18\frac{€}{\text{año}}}{1{,}767,12\frac{\text{horas}}{\text{año}}} = 17,58\frac{€}{\text{hora}} \tag{9.2}$$

[1]Salario convenio Euros/mes por 14 pagas técnico 1643,98 €[27]

En la siguiente tabla (tabla 9.3)se muestra una distribución temporal aproximada del trabajo del total de dos 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 estudio previo	35	horas
Estudio del problema	50	horas
Desarrollo de la solución adoptada	100	horas
Depuración	70	horas
Documentación	100	horas
Total de horas empleadas	355	horas

Tabla 9.3: *Distribución temporal de trabajo*

Para finalizar calcularemos 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:

$$355 \quad \text{horas} \cdot 17,58 \frac{€}{\text{hora}} = 6{,}240,9€ \tag{9.3}$$

COSTE PERSONAL DIRECTO	**6.240,9 €**

9.2.2. Costes de amortización de equipos y programas informáticos

En este apartado están incluidos los costes de uso de los diferentes equipos y programas usados que tienen una vida útil mayor que la del presente proyecto. Quedan excluídos el uso de sistemas operativos y programas Open Source tales como Ubuntu 14.04, Open Office Writer, ROS y librerías afines, puesto que no suponen coste alguno por ser de libre disposición.

Para hacer este cómputo calcularemos su coste total y su tiempo de amortización, que es la vida útil estimada del correspondiente material. A continuación, se le aplicará a dicho resultado un factor de corrección que depende del número de años que se haya considerado como tiempo de amortización:

- Ordenador sobremesa HP compaq: vida útil: 6 años. Factor de corrección por amortización 0.23.

- Microsoft Windows Vista Professional. Renovación y actualización de sistemas operativos por parte de Microsoft: 3 años. factor de corrección para la amortización 0.3.

Haciendo aplicación al precio total de los productos, queda la tabla 9.4:

MATERIAL	IMPORTE	FACTOR	AMORTIZACIÓN ANUAL
Microsoft Windows Vista Profess.	134,34 €	0,3	40,30 €
Ordenador sobremesa HP Compaq	600,00 €	0.23	138,00 €
Total material	**734,34 €**		**178,30 €**

Tabla 9.4: *Amortizaciones material*

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

$$\frac{178,30\frac{€}{\text{año}}}{1{,}767,12\frac{\text{horas}}{\text{año}}} = 0,1\frac{€}{\text{hora}} \tag{9.4}$$

Finalmente, para hallar 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, a lo largo de todas las etapas de desarrollo del proyecto:

$$355 \quad \text{horas} \cdot 0,1\frac{€}{\text{hora}} = 35,5€ \tag{9.5}$$

COSTES DE AMORTIZACIÓN DE RECURSOS OFICINA	35,5 €

9.2.3. Costes de equipamiento físico

En estos costes se incluyen tanto el coste de los elementos sensores utilizados en el sistema de visión artificial para este proyecto, que principalmente son dos, la Microsoft Kinect y el láser SICK LMS100, como los servomotores (actuadores) y la placa driver que les gobierna.

MATERIAL	IMPORTE
Microsoft Kinect	39,00 €
Láser SICK LMS100	3.738,00 €
Servomotores (x6)	64,00 €
Controladora ssc32	34,54 €
Total equipamiento físico	**3.875,54 €**

Tabla 9.5: *Costes del equipo físico*

COSTES DE EQUIPAMIENTO FÍSICO	3.875,54 €

9.2.4. Costes directos totales

Los costes directos totales, que son la suma de los costes directos parciales anteriores, son:

$$6{,}240{,}9€ + 35{,}5€ + 3{,}875{,}54€ = 10{,}151{,}94€ \tag{9.6}$$

COSTES DIRECTOS	10.151,94 €

9.3. Costes indirectos

Los costes indirectos se refieren a aquellos gastos derivados de la actividad asociada al proyecto, pero que no se pueden asignar directamente al mismo porque pueden estar involucrados en otros trabajos o proyectos. Los costes indirectos del presente proyecto incluyen diferentes consumos y gastos de gestión administrativa.

PARTIDAS DE COSTES INDIRECTOS

Gestión administrativa	145,00 €
Consumo eléctrico	120,00 €
Consumos por desplazamiento	55,00 €
Total de gastos indirectos	**320,00 €**

Tabla 9.6: *Costes Indirectos*

COSTES INDIRECTOS	320,00 €

9.4. Aproximación al coste total del proyecto

Los costes totales son el resultado de la suma de los gastos directos e indirectos, siendo el montaje total para este proyecto:

COSTES TOTALES

Costes directos	10.151,94 €
Costes indirectos	320,00 €
Coste total del proyecto	**10.471,94 €**

Tabla 9.7: *Costes Totales*

COSTE TOTAL DEL PROYECTO	**10.471,94 €**

Capítulo 10

CONCLUSIÓN Y FUTURAS LÍNEAS DE DESARROLLO

En este último capítulo es importante hacer una recapitulación del proyecto desarrollado, revisando los objetivos alcanzados, los logros que se han conseguido y la revisión de líneas de investigación que podrían aplicarse en el futuro sobre la base del proyecto realizado.

El objetivo principal de este proyecto era el de dotar al robot Sacarino II de una mayor expresividad, para lograr un mejor impacto en la interacción hombre máquina. De los estándares de comportamiento social se deduce que es importante, a la hora de tratar con una persona, mirarla a los ojos, signo de estar activo en la conversación; y por otra parte, la gesticulación y expresividad facial acentúan el acercamiento y lo hacen más humano, reforzando positivamente la interacción.

Este proyecto ha seguido una consecución de etapas a lo largo del tiempo de desarrollo. En una primera fase del proyecto se ha tenido que realizar un establecimiento de objetivos, análisis de la tecnología actual, los elementos de sensorización y, en general, el equipo del que se disponía.

Seguidamente, se ha dedicado una segunda etapa a la familiarización con el robot, la comprensión de su mecánica y, sobre todo, el desempeño con el sistema operativo en el que está desarrollado, ROS.

Durante la tercera fase se han formulado diversas aproximaciones al problema. La solución escogida para satisfacer los objetivos se apoya en dos módulos, uno de seguimiento facial y un segundo de expresiones.

El primer módulo realiza la detección de usuarios mediante un láser SICK. Una vez seleccionado uno de los usuarios se realiza su seguimiento visual mediante sensor Kinect integrado en la propia cabeza. Este seguimiento realiza de forma suave evitando oscilaciones mediante la inclusión de un controlador no lineal sigmoidal. Los experimentos han demostrado un comportamiento natural en el seguimiento de usuarios.

Por otra parte, el subsistema de expresión gestual basado en FACS, proporciona al robot una gran expresividad, y una experiencia que resulta agradable para el usuario. Para coordinar estos dos subsistemas, la biblioteca actionlib de ROS se ha mostrado como una herramienta muy útil a la hora de proporcionar al secuenciador del robot una realimentación en tiempo real de las órdenes que éste envía al sistema de interacción.

Del análisis de ambos módulos se desprende una mejora significativa de la expresividad y fluidez del robot. Esto tiene un efecto positivo en la interacción Hombre-Máquina, creando una sensación de familiaridad y cercanía con el usuario.

Se concluye que, a la vista de los resultados obtenidos, el robot se muestra adecuado para plantear su integración en un entorno operacional donde interactúe con todo tipo de personas, y, en particular, con personas mayores.

10.1. Líneas futuras

A partir del desarrollo propuesto en este proyecto y como líneas futuras de investigación se propone por un lado, la posible incorporación de un sistema de captación de estímulos externos que complemente al sistema de interacción desarrollado. Un ejemplo puede ser la imitación de la expresión facial del usuario del usuario con el que se esté interactuando.

Por otra parte, se propone la incorporación de elementos adicionales de apoyo a la interacción como pueden ser sean brazos o piernas.

Finalmente, cabe decir que un robot social es un sistema complejo. Este proyecto aborda solamente el sistema de interacción, pero un robot social está compuesto por muchos más módulos, entre otros, navegación, gestores de diálogo, localización (SLAM), y otros servicios. Por ello, cualquier sistema que se añada o investigue forma parte de las líneas futuras de desarrollo del presente proyecto.

10.2. Publicaciones relacionadas

Como complemento a la labor desarrollada en este proyecto la siguiente publicación

Domínguez M., Marcos S., Zalama E., García-Bermejo J. "Sistema de interacción visual para un robot social"

ha sido enviada a las Jornadas de Automática 2017, a celebrar en Gijón del 6 al 8 de Septiembre. En dicha publicación se expone un resumen del contenido tratado en este proyecto. Esto puede resultar de gran interés para otros investigadores e ingenieros que desarrollen su labor en el ámbito de la robótica social.

APÉNDICES

APÉNDICE A

DIAGRAMA DE GANTT

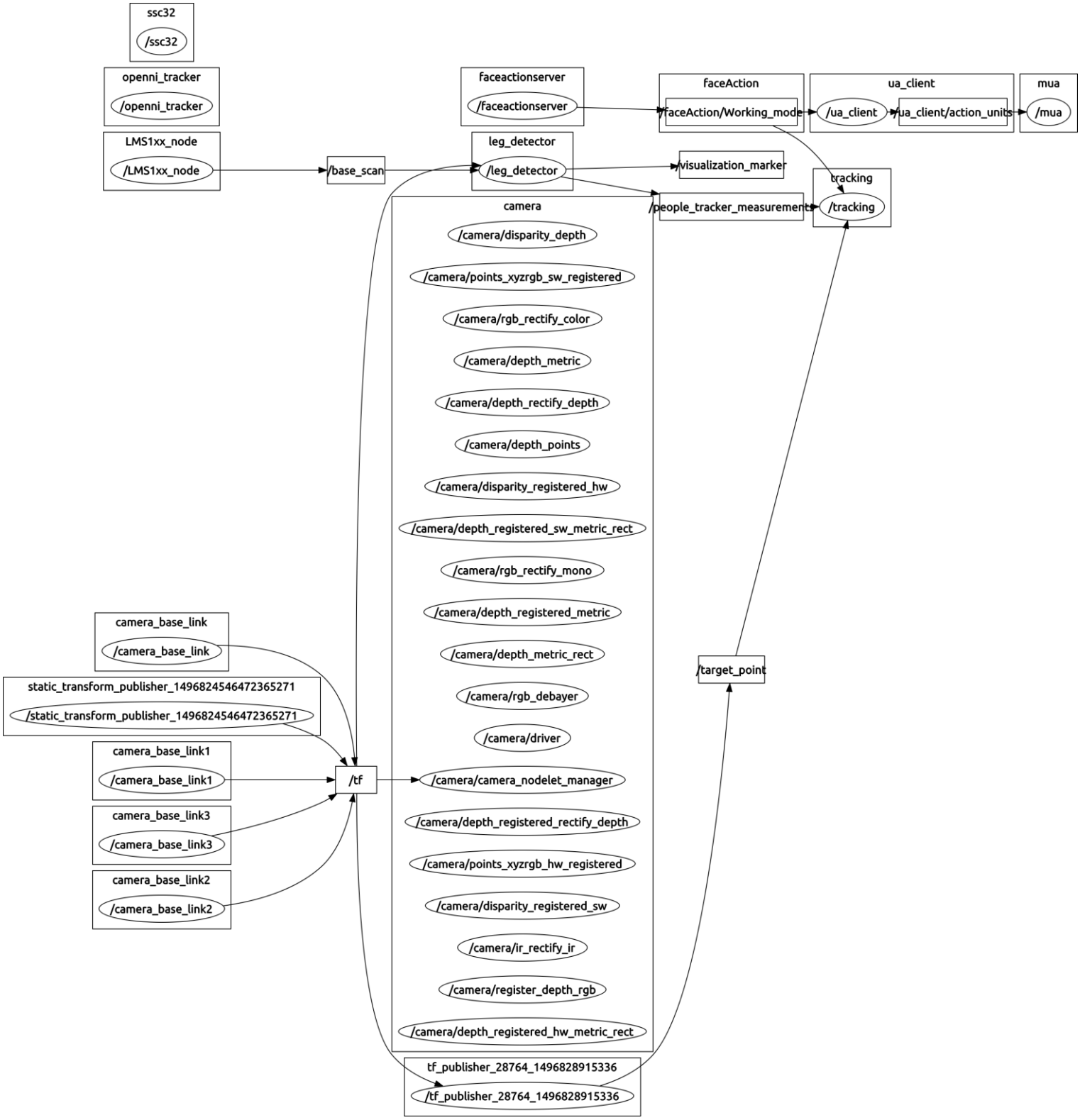
En este apéndice queda reflejado el diagrama de Gantt del presente proyecto, con la distribución temporal de las tareas llevadas a cabo.


Febrero
Marzo
Abril
Mayo
Junio
Semana 2
Semana 3
Semana 4
Semana 1
Semana 2
Semana 3
Semana 4
Semana 1
Semana 2
Semana 3
Semana 4
Semana 1
Semana 2
Semana 3
Semana 4
Semana 1
Semana 2
Establecimiento de objetivos
FASE I
Análisis de la tecnología
Equipo disponible
FASE II
Estudio del robot
Familiarización con ROS
FASE III
Susbistema de seguimiento
Subsistema de expresiones
Interfaz gráfica
Sistema Global de interacción
FASE IV
Depuración

APÉNDICE B

GRAFO GLOBAL DE NODOS

Apéndice dedicado a presentar, en forma de grafo, la estructura del sistema de interacción.


ssc32
/ssc32
openni_tracker
/openni_tracker
LMS1xx_node
/LMS1xx_node
/base_scan
faceactionserver
/faceactionserver
faceAction
/faceAction/Working_mode
ua_client
/ua_client
/ua_client/action_units
mua
/mua
leg_detector
/leg_detector
/visualization_marker
/people_tracker_measurements
tracking
/tracking
camera
/camera/disparity_depth
/camera/points_xyzrgb_sw_registered
/camera/rgb_rectify_color
/camera/depth_metric
/camera/depth_rectify_depth
/camera/depth_points
/camera/disparity_registered_hw
/camera/depth_registered_sw_metric_rect
/camera/rgb_rectify_mono
/camera/depth_registered_metric
/camera/depth_metric_rect
/camera/rgb_debayer
/camera/driver
/camera/camera_nodelet_manager
/camera/depth_registered_rectify_depth
/camera/points_xyzrgb_hw_registered
/camera/disparity_registered_sw
/camera/ir_rectify_ir
/camera/register_depth_rgb
/camera/depth_registered_hw_metric_rect
camera_base_link
/camera_base_link
static_transform_publisher_1496824546472365271
/static_transform_publisher_1496824546472365271
camera_base_link1
/camera_base_link1
camera_base_link3
/camera_base_link3
camera_base_link2
/camera_base_link2
/tf
/target_point
tf_publisher_28764_1496828915336
/tf_publisher_28764_1496828915336

APÉNDICE C

HOJA DE CARACTERÍSTICAS SICK

En este apéndice se presentan las características del sensor láser SICK LMS 100 usado en este proyecto.

Ordering information

Type	Part no.
LMS100-10000	1041113

Other models and accessories ➜ www.sick.com/LMS1xx

Detailed technical data

Features

Field of application	Indoor
Light source	Infrared (905 nm)
Laser class	1 (IEC 60825-1 (2007-3))
Aperture angle	270°
Scanning frequency	25 Hz / 50 Hz
Angular resolution	0.25° 0.5°
Heating	No
Operating range	0.5 m ... 20 m
Max. range with 10 % reflectivity	18 m
Amount of evaluated echoes	2
Fog correction	Yes

Performance

Response time	≥ 20 ms
Detectable object shape	Almost any
Systematic error	± 30 mm [1)]
Statistical error	12 mm [1)]
Integrated application	Field evaluation with flexible fields
Number of field sets	10 fields
Simultaneous evaluation cases	10

[1)] Typical value; actual value depends on environmental conditions.

Interfaces

Ethernet	✓, TCP/IP
Remark	OPC
Function	Host
Data transmission rate	10/100 MBit/s

Serial	✓, RS-232
Function	Host, AUX
Data transmission rate	9.6 kBaud ... 115.2 kBaud
CAN	✓
Function	Extension of outputs
Switching inputs	2 digital and 2 encoder inputs
Switching outputs	3
Optical indicators	7-segment display (plus 5 LEDs showing device status, contamination warning and initial condition)

Mechanics/electronics

Electrical connection	1 x system plug with screw terminal block
Operating voltage	10.8 V DC ... 30 V DC
Power consumption	Typ. 8 W
Housing color	Light blue (RAL 5012)
Enclosure rating	IP65 (EN 60529, Section 14.2.5)
Protection class	III (EN 50178 (1997;10))
Weight	1.1 kg
Dimensions (L x W x H)	105 mm x 102 mm x 152 mm

Ambient data

Object remission	2 % ... > 1,000 % (reflectors)
Electromagnetic compatibility (EMC)	EN 61000-6-2:2005 / EN 61000-6-4 (2007-01)
Vibration resistance	EN 60068-2-6 (1995-04)
Shock resistance	EN 60068-2-27 (1993-03)
Ambient operating temperature	0 °C ... +50 °C
Storage temperature	–30 °C ... +70 °C
Ambient light immunity	40,000 lx

General notes

Note on use	The sensor does not constitute a safety component as defined by relevant legislation on machine safety.

Classifications

ECl@ss 5.0	27270990
ECl@ss 5.1.4	27270990
ECl@ss 6.0	27270913
ECl@ss 6.2	27270913
ECl@ss 7.0	27270913
ECl@ss 8.0	27270913
ECl@ss 8.1	27270913
ECl@ss 9.0	27270913
ETIM 5.0	EC002550
ETIM 6.0	EC002550
UNSPSC 16.0901	46171620

APÉNDICE D

HOJA DE CARACTERÍSTICAS KINECT

En este apéndice se detallan las características del sensor kinect usado para este proyecto.

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

APÉNDICE E

CÓDIGOS FACS

Apéndice dedicado a exponer las tablas de codificación del estándar FACS mencionadas en el capítulo 6.

Códigos Principales

Número de acción	Nombre de acción
0	Rostro Neutral
1	Levantamiento interior de ceja
2	Levantamiento exterior de ceja
4	Bajar cejas
5	Levantamiento del párpado superior
6	Levantamiento de mejillas
7	Apretar parpado(s)
8	Labios encimados uno de otro
9	Arrugar la nariz
10	Levantamiento del labio superior
11	Profundidad nasolabial
12	Tiramiento labial esquinal
13	Tiramiento labial frontal
14	Hoyuelo facial
15	Depresión labial esquinal
16	Depresión labial frontal
17	Levantamiento de mejillas
18	Arruga labial
19	Muestro de lengua
20	Apretar los labios
21	Apretamiento de cuello
22	Embudo labial
23	Morder labios

Número de acción	Nombre de acción
24	Presión Labial
25	Deslizamiento labial
26	Caída de la mandíbula
27	Apretamiento bucal
28	Lamido labial
29	Tracción de la mandíbula
30	Deslizamiento de mandíbula
31	Contracción mandibular
32	Mordida labial
33	Succión de mejillas
34	Inflar mejillas
35	Soplido de mejillas
36	Protuberancia de lengua
37	Limpieza labial
38	Dilatado nasal
39	Compresión nasal
41	Bajeza de ojeras
42	Contracción retinal
43	Ojos cerrados
44	Recolector retinal
45	Parpadeo
46	Guiño

Códigos de motores de cabeza

Número de acción	Nombre de acción
51	Girar cabeza a la izquierda
52	Girar cabeza a la derecha
53	Alzar la cabeza
54	Bajar la cabeza
55	Inclinación de la cabeza hacia la derecha
M55	Inclinación de la cabeza a la izquierda
56	Leve inclinación
M56	Inclinación derivada en el cuello
57	Head Forward
M57	Empujar cabeza hacia adelante
58	Empujar hacia atrás
M59	Agitar cabeza hacia arriba y abajo
M60	Agitar cabeza de lado a lado
M83	Inclinación diagonal

Códigos de motores de visibilidad

Número de acción	Nombre de acción
70	Frente y cejas no visibles
71	Ojos no visibles
72	Mentón no visible
73	Rostro sin movimiento
74	Indescifrable

Códigos de movimiento ocular

Número de acción	Nombre de acción
61	Mover ojos hacia la izquierda
M61	Ojos a la izquierda
62	Mover ojos a la derecha
M62	Ojos a la derecha
63	Ojos hacia arriba
64	Ojos hacia abajo
65	Leucoma
66	Estrabismo
M68	Deslizamiento ocular
69	Fijar los ojos en otra persona
M69	Posicionamiento ocular y frontal

Bibliografía

[1] Masahiro Mori. The uncanny valley. 1970.

[2] Tamara Botrán Herrero. Diseño de un robot social interactivo. 2015.

[3] Roberto Pinillos Herrero. Diseño de un robot social y validación de servicios en entornos hoteleros. 2014.

[4] Carlo Dal Mutto et al. Time-of-flight cameras and microsoft kinect. page 51, 2008.

[5] Andrea Fossati et al. Consumer depth cameras for computer vision. pages 5–25, 2013.

[6] D. Loza. Diseño y construcción de una cabeza mecatrónica de aspecto realista. 2013.

[7] Online, Visitado el 20 de Febrero de 2017 `http://www.etitudela.com/profesores/rpm/rpm/downloads/robotica.pdf`.

[8] Roberto Pinillos Herrero. Diseño de un robot social y validación de servicios en entornos hoteleros. 2014.

[9] Alexis Juárez Sánchez. Desarrollo de un sistema de visón-3d para su integración en un robot móvil social. 2014.

[10] Mª Guadalupe Sánchez Escribano. *Sistema de visión para un robot social.* PhD thesis, Universidad Politécnica de Madrid, 2012.

[11] T.Sheridan. Eight ultimate challenges of humanrobot communication, in: Proceedings of the international workshop on robots and human communication. 1997.

[12] Roball F.Michaud, S. Caron. An autonomous toy-rolling robot, in: Proceedings of the workshop on interactive robot entertainment. 2000.

[13] Hanson D. Olney A. Pereira I. A. Zielke M. Upending the uncanny valley. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 4*, pages 1728–1729, 2005.

[14] Roberto Pinillos Herrero Samuel Marcos Pablos Raúl Feliz Eduardo Zalama Jaime Gómez García-Bermejo. Long-term assesment of a service robot in a hotel environment. 2016.

[15] S. Dellaerty F. Foxy, D. Burgardz W. Thrun. Montecarlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA-99)*, 1999.

[16] L. Ladicky P. Sturgess C. Russell S. Sengupta Y. Bastanlar W. Clocksin and P. Torr. Joint optimisation for object class segmentation and dense stereo reconstruction. pages 1–12, 2011.

[17] J. Leens S. Pirard O. Barnich M. Van Droogenbroeck and J.-M. Wagner. Combining color, depth, and motion for video segmentation. pages 104–113, 2009.

[18] J. Shi and J. Malik. Normalized cuts and image segmentation. ieee transactions on pattern analysis and machine intelligence. 2000.

[19] K.: Khoshelham. Acuracy analysis of kinect depth data. 2011.

[20] `http://www.ros.org/`.

[21] Online, Visitado el 15 de Febrero de 2017 `https://github.com/ros-drivers/openni_tracker/issues/9`.

[22] Online, Visitado el 4 de Abril de 2017 `http://wiki.ros.org/leg_detector`.

[23] P. Ekman and W. Friesen. *Facial Action Coding System: A Technique for the Measurement of Facial Movement.* Consulting Psychologists Press.

[24] lentin Joseph. *mastering ROS for robotics programming.* PACKT Open Source.

[25] Online, Visitado el 10 de Marzo de 2017 `http://docs.wxwidgets.org/3.0/page_introduction.html`.

[26] Online, Visitado el 7 de Marzo de 2017 `https://wxpython.org/what.php`.

[27] Online, Visitado el 6 de Mayo de 2017 `https://www.boe.es/boe/dias/2017/01/18/pdfs/BOE-A-2017-542.pdf`.

[28] Julio Molleda Meré. *Técnicas de visión por computador para la reconstruccnión en tiempo real de la forma 3D de productos laminados.* PhD thesis, Universidad de Oviedo, 2012.

[29] Online, Visitado el 25 de Marzo de 2017 https://es.scribd.com/doc/259914658/Sistemas-de-Vision-Artificial-Historia-Componentes-y-Procesamiento-de-Imagenes.