Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA INGENIERÍAS INDUSTRIALES

MÁSTER INVESTIGACIÓN EN INGENIERÍA DE PROCESOS Y SISTEMAS

TRABAJO FIN DE MÁSTER

DISEÑO DE UN ROBOT HUMANOIDE DE BAJO COSTE MEDIANTE ROBOT OPERATING SYSTEM

Autor: MARIO DOMÍNGUEZ LÓPEZ

Tutor: DR. EDUARDO ZALAMA CASANOVA

25 de Junio de 2019

Agradecimientos

En lo personal, a mis padres María del Mar, José Luis y mi hermana Raquel, que son pilares fundamentales para cualquier proyecto. A mi tutor Eduardo Zalama, por la oportunidad, guía y ayuda prestada. A Dios, por estar siempre a mi lado.

Nunca andes por el camino trazado, pues te conducirá a donde otros ya fueron.

Alexander Graham Bell

Resumen

En la actualidad son muchos los esfuerzos de investigación en el campo de la robótica humanoide bípeda, con la motivación de dotar a los robots de capacidades y habilidades lo más cercanas a la naturaleza humana. Son varias las áreas de ingeniería involucradas en este tipo de robótica, como son la mecánica, el control, la electrónica, los sistemas de tiempo real o la visión artifical.

El presente proyecto comprende el diseño y desarrollo del robot Clank, que pretende ser una plataforma robótica humanoide bípeda de bajo coste para la investigación de este tipo de sistemas. Para abordar el diseño se hará uso de la tecnología de impresión aditiva, mientras que el desarrollo software centra en tres áreas: la visión artificial, el control de balanceo y la coordinación postural. El uso de librerías de libre acceso y ROS son el entorno de trabajo elegido para dar soporte a todo el sistema, y, en concreto, la integración del estándar ROS Industrial.

Palabras clave: Robot humanoide, ROS Industrial, Interacción humano-robot

Abstract

Nowadays, a lot of research efforts are made in the field of bipedal humanoid robotics, with the motivation of endowing robots with the most similar skills and capabilities to human nature. A wide range of engineering areas are involved in this kind of robotics: mechanics, control, electronics, real-time systems or artificial vision.

The present proyect addresses the design and development of Clank, which pretends to be a low-cost bipedal humanoid platform to explore this kind of systems. In the design, additive 3D printing will be used, whereas the software development will be focused on three areas: artificial vision, balance control, and postural coordination. Open source libraries and ROS are the chosen framework to give support to all the system, and, more specifically, the integration of the ROS Industrial standard.

Keywords: Humanoid robot, ROS Industrial, Human Machine interaction

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 y Justificación

El presente proyecto se enmarca dentro del ámbito de la robótica, sistemas embebidos, visión artificial e impresión 3D, que juegan actualmente un papel fundamental y emergente en el desarrollo tecnologico. Dentro de la robótica se distinguen, al menos, dos categorías:

- La robótica industrial: enfocada sensiblemente a procesos de producción y mundo empresarial. En ella, el robot desempeña tareas repetitivas sustituyendo al operador humano, siempre menos preciso que el propio robot.
- 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. El objetivo final es que el robot sirva a las personas; que pueda proporcionar servicios como información o ayuda en el hogar, hospitales, hoteles etc.

Aunque esta división no es rígida, 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 [2].

El auge de la robótica de servicios ha sido significativo en los últimos años, sirvan como ejemplo el robot Pepper de SoftBank Robotics [1] o el robot Roomba. Dentro de la robótica de servicios, uno de los campos que está en actual desarrollo es el de la robótica humanoide, que presenta importantes ventajas como el aporte de piernas frente a ruedas y que no limita la operación a superficies planas, escaleras o zonas irregulares.

Por otro lado, la robótica humanoide entraña varias dificultades y retos como son:

- Localización (modelo cinemático complejo) y navegación.
- Sincronización de articulaciones.
- Control de equilibrio.
- Interacción con el entorno, como por ejemplo, el agarre de objetos.

Por lo tanto, para su abordaje es necesaria la aplicación de diversas disciplinas como la mecánica, electrónica, computación, control y percepción, además de un elevado coste económico. Es por ello que la fabricación aditiva (impresión 3D) ha permitido la construcción de prototipos más económicos. Existen numerosos tipos de filamentos y materiales, desde la impresión en plásticos como PLA o ABS hasta la impresión metálica en la que se incluyen partículas suspendidas de metales en un lecho fluido.

El desarrollo de la electrónica ha dado lugar a microcontroladores de mayores prestaciones, potenciando el control distribuido y aumentando el número de sensores capaces de procesar, todo ello, en un tamaño reducido. Una consecuencia de esto, es que hoy en día existen sistemas embebidos de bajo coste, como RaspBerry Pi, Arduino, Odroid... capaces incluso de soportar sistemas operativos basados en Linux, como Ubuntu o Debian.

Por otro lado, el desarrollo de microprocesadores más potentes, ha permitido una mejora significativa en el procesamiento de imágenes. Esto hace que hoy en día, la visión artificial sea una tecnología multidisciplinar, aplicable en diversos sectores. Dentro de

[1] https://www.softbankrobotics.com/emea/en/pepper

la visión artificial podemos distinguir, al menos, dos campos, el primero sería la visión 2D en imagen donde habitualmente se utilizan técnicas de reconocimiento de patrones basadas en métodos por aprendizaje o técnicas más deterministas de extracción de características como contornos, o ciertos patrones. El segundo campo sería la visión 3D y tratamiento de nubes de puntos, ya sean obtenidos por triangulación pasiva (stereo) o activa (luz estructurado, por ejemplo) en el cada vez existen más sensores disponibles, como por ejemplo, el Velodyne, utilizado en conducción autónoma.

1.2. Objetivos

A continuación se plantean los objetivos generales y específicos del proyecto.

El objetivo general del presente proyecto es el de la creación de una plataforma robótica humanoide bípeda de bajo coste integrada bajo el sistema operativo robótico (ROS) que esté dotada de un control de movimiento e interacción humano-robot aceptable.

Dentro de este gran objetivo, se encuentran aspectos relacionados con la correcta elección del software (elección de librerías de trabajo) y hardware (correcta selección de elementos sensores y actuadores). Esto resulta en los siguientes objetivos específicos:

- Ser una plataforma de bajo coste mediante técnica de impresión aditiva, con capacidad de movimiento y desplazamiento con al menos 18 grados de libertad.
- Que el robot sea capaz de planificar y ejecutar trayectorias en sus articulaciones y mantener un control postural.
- Lograr que el robot sea capaz de mantener el equilibrio ante perturbaciones en el plano de cabeceo.
- Que el robot tenga la habilidad de seguimiento facial («tracking») de la persona con la que está interaccionando, apoyando y mejorando la interacción.

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, entre los que destaca, la adopción del estándar ROS::Industrial como enfoque de desarrollo y control.

Clank es el nombre que recibe el robot social desarrollado en este proyecto, y es un robot humanoide de bajo coste cuyo objetivo es el de ser una plataforma de investigación donde abordar aspectos relacionados con la interacción humano-robot, profundizar en el estudio de los robots humanoides bípedos, todo esto integrado en un mismo entorno de desarrollo. Otros trabajos enfocados sobre este mismo tema han sido tenidos en cuenta [3], [4] [5] y han servido como apoyo para la realización de este proyecto.

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, más concretamente, se profundizará en la robótica humanoide haciendo un estudio de los robots referentes en este ámbito.

A lo largo del tercer capítulo se hablará de aspectos relacionados con el hardware del robot desarrollado como son la mecánica y componentes electrónicos elegidos. A continuación, se adentrará en el segundo bloque de la memoria, en el cuarto capítulo se tratará la explicación del entorno de desarrollo software elegido para gobernar la plataforma y el sistema de planificación de trayectorias y control postural.

Seguidamente, el quinto capítulo, referido principalmente al sistema de estabilidad, el fundamento de la sensórica elegida, tratamiento de la señal y el lazo de control.

Para cerrar este segundo bloque, se abordará el funcionamiento del sistema de seguimiento facial, las librerías y el algoritmo utilizados.

En este punto se entrará el tercer bloque, que comprende los capítulos 7, 8 y 9. En el séptimo capítulo se desarrolla una aproximación económica al coste real de la realización del proyecto, con el consiguientes desglose por costes de materiales y recursos. Finalmente, los capítulo que cierran la presente memoria están destinados a conclusiones, aprendizajes, y futuras líneas de desarrollo e investigación.

Capítulo 2

ROBÓTICA SOCIAL Y HUMANOIDE

Este capítulo profundiza en el concepto de robótica social y robótica humanoide bípeda además de analizar el estado del arte en este momento.

2.1. 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 [6] 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.

2.2. 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 [7]. Resulta evidente que unos robots tienen una realización más compleja que otros, véase si no la diferencia entre *Asimo* (Honda) e *IO*, figura 2.1.

(a) Asimo (b) IO

Figura 2.1: Complejidad en los robots sociales

La forma y estructura del robot (figura 2.1) 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 (figura 2.1 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,[8].

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 predisposi-

ción 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.

2.2.1. 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.

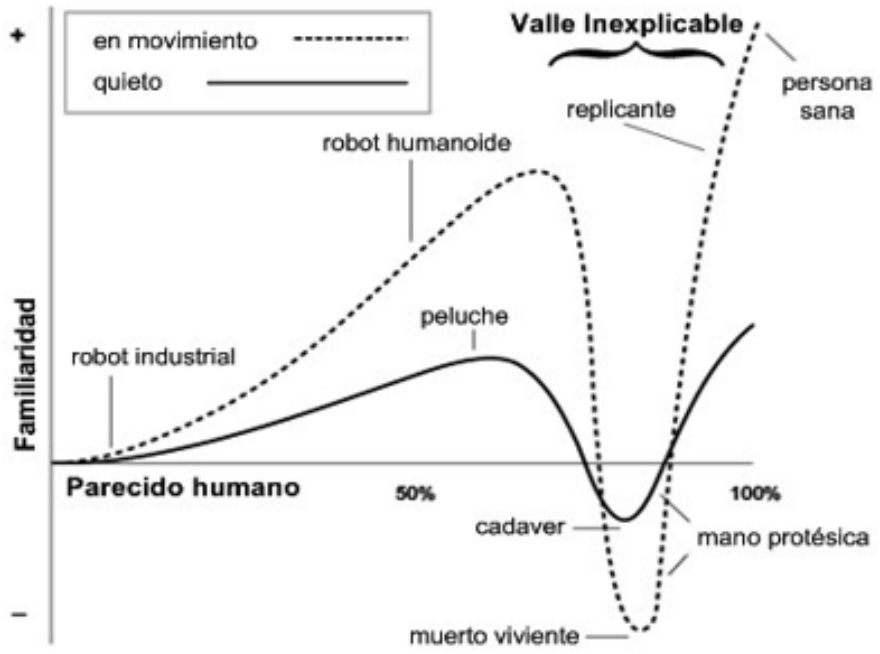


Figura 2.2: El valle inexplicable de Mori (de [1])

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[6].

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 [9].

2.3. Robótica humanoide

El Robot Institute of America define a un robot como: «maquina de manipulación automática, reprogramable y funcional que puede posicionar y orientar materiales, piezas, herramientas o dispositivos especiales para la ejecución de diversas tareas, ya sea en una posición fija o en movimiento». Sin embargo al hacer referencia de un robot humanoide, se considera a cualquier robot que presentan una apariencia similar a la humana: 2 piernas, 2 brazos, tronco, cabeza, etc. Dentro de la investigacion de la robótica humanoide, se busca dotar de autonomía a cada robot, buscando simular de la manera mas precisa posible el comportamiento de un ser humano.

En la industria, la llamada nueva revolución industrial o Industria 4.0 propone como uno de sus objetivos principales la introducción de robots más flexibles que puedan realizar distintas tareas en colaboración con otros robots y con humanos de forma segura y sin necesidad de mantenerlos en entornos o celdas aisladas. De esta forma, los robots colaborativos se convierten en una herramienta clave, controlada por el operador, que podrá hacer uso de ellos en tareas peligrosas, repetitivas o que requieran de un mayor esferzo físico. Mientras tanto, a persona puede centrarse en la toma de decisiones más

complejas y en la parte más humana del trabajo.

La ventaja principal de la utilización de robots humanoides en entornos indutriales es que este tipo de robots puede trabajar directamente en el mismo entorno que los humanos, sin que sea necesario introducir modificaciones sobre este. Además, como a mayoría de utensilios, maquinarias y escenarios están adaptados para el uso humano, también lo estarán para el uso de robots humanoides.

Otro aspecto a tener en cuenta es que el medio de tracción de los robots humanoides se puede adaptar al entorno en el que vayan a ser utilizados. De este modo, si el entorno en el que se quiere incorporar no contiene obstáculos y el suelo es liso, se puede utilizar un robot humanoide que se desplace utilizando ruedas, mientras que si se trata de un entorno real (escaleras, suelo con desperfectos, etc) pueden utilizarse humanoides bípedos[1].

Para analizar el estado del arte de la robótica humanoide bípeda en términos de prototipos existentes se proponen los siguientes ejemplos:

Atlas, Boston Dynamics

Figura 2.3: Atlas

[1]Zoe Falomir Llansola, Universitat Jaume I, Campus de Riu Sec, Castellón

Atlas es un robot humanoide bípedo desarrollado por la compañía Boston Dynamics, con la financiación de la *Defense Advanced Research Projects Agency, DARPA*. El robot mide 1,80 metros y su objetivo principal es la realización de diferentes tareas de búsqueda y rescate. El modelo mostrado en 2.3 se dió a conocer en Julio 2013.

Vakyrie, NASA

Figura 2.4: Valkyrie

Valkyrie es un robot desarrollado por NASA para competir en 2013 en el DARPA Robotics Challenge (DRC) trials. Este robot es completamente eléctrico y se ha diseñado para operar en entornos peligrosos para las personas. Este robot mide 1,80 metros y pesa 130 kg.

Talos, PAL-Robotics

Figura 2.5: Talos

Talos es la solución humanoide bípeda de PAL Robotics en 2019. Se trata de un robot totalmente eléctrico. Este robot está preparado para enfrentarse a tareas industriales complejas con sus 6Kg de carga en cada mano. Mide 1,75 metros y pesa 95 kg.

En resumen, la flexibilidad y adaptabilidad que aportan los robots humanoides justifica el crecimiento de este tipo de robots a escala internacional. Pueden atravesar zonas inalcanzables para el ser humano, son un apoyo para cualquier tarea ya sea de manipulación o transporte. Sin embargo son muchos los retos que se tienen que superar, desde dotarles de inteligencia artificial para toma de decisiones a la investigación en sistemas de tiempo real se presenta como fundamental para el control y monitorización de toda la percepción y actuación del robot, de manera que se asegure su respuesta ante tareas o situaciones críticas.

Además, entre otros retos se incluyen la mejora de la interfaz social a través del razonamiento y pensamiento y, a nivel hardware, pese a que los sensores han llegado a un buen nivel de maduración, se requiere investigar en nuevos materiales que ayuden a optimizar el consumo de energía.

Todo esto se presenta como los proximos retos a presente y futuro que encara la robótica humanoide.

Capítulo 3

EL ROBOT HUMANOIDE CLANK

El presente capítulo está dedicado a la descripción de Clank 3.1, el robot humanoide desarrollado en este proyecto. Se revisarán aspectos mecánicos, electrónicos y de desarrollo de sofware. Clank tiene un peso de 1.6 Kg y mide 40 cm.

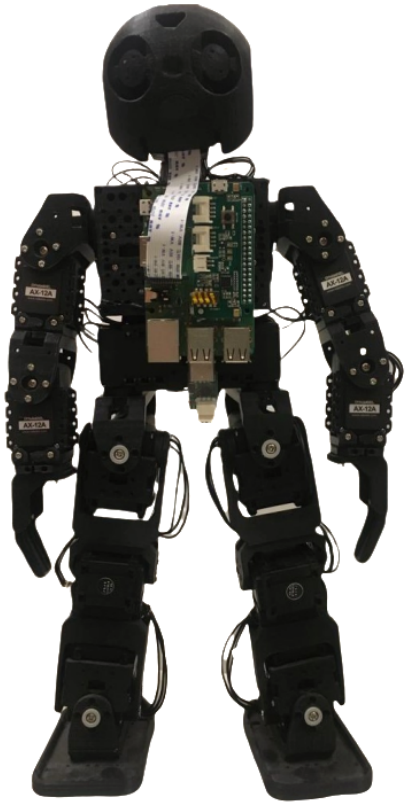


Figura 3.1: Clank

Para el cumplimiento del objetivo principal de este proyecto se ha pensado en la creación de una plataforma con gran movilidad, modular, y en la que puedan integrar distintas áreas como visión artificial, planificación y control. Además, el robot tiene que tener un tamaño manejable y suficientemente ligero para que los actuadores que se elijan tengan capacidad suficiente para mover el robot.

A continuación se profundiza en los aspectos relacionados con el hardware y software necesario para cumplir con las especificaciones fijadas.

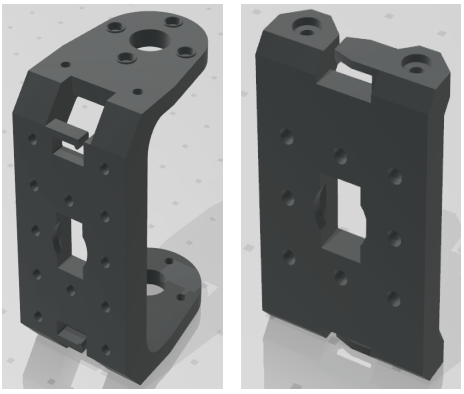
3.1. Desarrollo hardware

3.1.1. Impresión 3D. Componentes y mecánica

Uno de los objetivos del proyecto es crear una plataforma de bajo coste, es por ello que el robot está construido en su totalidad en material PLA. Todas las pizas de Clank (figura 3.1) han sido creadas con la idea de que su uso sea flexible a la hora de combinarlas para crear distintos tipos de robots, desde humanoides bípedos hasta hexápodos y están basadas en el proyecto Bioloid de ROBOTIS[1].

Todos las piezas del cuerpo del robot incluyen distintos agujeros de diferentes tamaños, esto es para desempeñar distintas funciones en las articulaciones, bien para acoplar unas piezas a otras, bien para acoplar la electrónica de a mejor manera y para aligerar el peso del robot puesto que tendrá un gran impacto en el rendimiento dinámico final.

[1] http://www.robotis.us/bioloid-1/



(a) F2, codo (b) F3, tobillo

Figura 3.2: Ejemplo de piezas modulares

Se pretendía que la cabeza del robot tuviese, al menos, un grado de libertad, por tanto para el diseño se ha tomado como referencia la del robot DarwinOp2 y se han logrado incluir hasta dos servomotores para ganar dos grados de libertad.

Como software de impresión se ha utilizado Cura Ultimaker[2] con los parámetros predefinidos para el modelo de impresora Anycubic Prusa i3.

3.1.2. Electrónica. Sensores, actuadores y controladores

Para que el robot Clank, y, en realidad, cualquier robot humanoide, interactúa de una manera aceptable con su entorno y sea capaz de tomar decisiones planificadas y reactivas, se deben incluir en el robot un conjunto sensor, actuador y de cálculo que resulten adecuados.

En cuanto a los sensores, el robot Clank cuenta, en primer lugar, con un **sensor Inercial** 3.3 (IMU, Acelerómetro, Giróscopo y magnetómetro) mini IMU v5 de Pololu que tiene las sigueintes características:

- Tamaño: 0.8 , 0.5 , 0.1 (pulgadas)
- Peso: 0.7 g.
- Interfaz: I2C.

[2]https://ultimaker.com/software/ultimaker-cura

- Ejes: pitch (x), roll (y), and yaw (z).
- Rangos de medida: +-125, +-245, +-500, +-1000, o +-2000 grados por s (gyro) +-2, +-4, +-8, or +-16 g (acelerómetro) +-4, +-8, +-12, or +-16 gauss (magnetómetro).

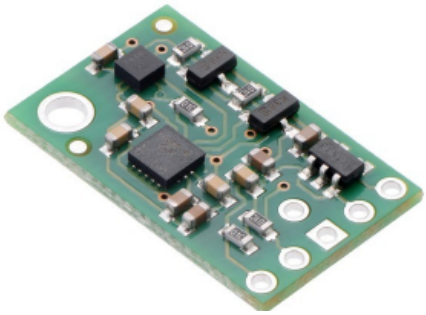


Figura 3.3: Mini IMU v5 pololu

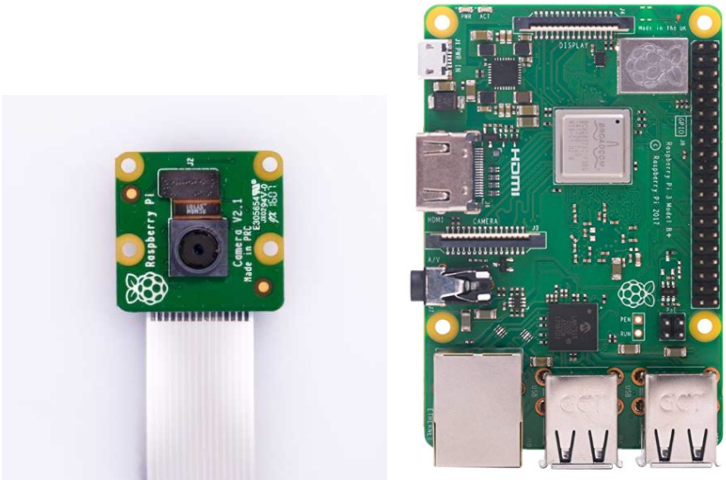
Este IMU será utilizado para retroalimentar la orientación del robot y para tomar sus datos por i2c se cuenta con un pequeño microcontrolador llamado OpenCM 9.04b, que toam los datos del i2c para pasarlos a interfaz serie en comunicación con el ordenador central, la Raspberry, por lo tanto, actuando como esclavo. OpenCm cuenta con un microprocesador ARM Cortex M3 de 32 bits y unos módulos entrada salida que soportan distintas intefaces como I2C, serie, SPI, UART y TTL.

Figura 3.4: OpenCM Robotis

También se ha utilizado una **cámara** ubicada en la cabeza, en el punto medio entre los ojos, de una resolución de 8MP con conexión mediante la interfaz SPI a su ordenador

central, una Raspberry Pi.

La Raspberry Pi modelo 3b+ es el último producto de la gama Raspberry Pi 3, con un procesador quad-core de 64 bits con 1,4 GHz, 1 GB LPDDR2 SDRAM , banda dual de 2,4 GHz y LAN inalámbrica de 5 GHz, Bluetooth 4.2 BLE, Ethernet más rápido y con capacidad para POE (*Power Over Ethernet*).



(a) Cámara Rpi 3 (b) Raspberry Pi 3b+

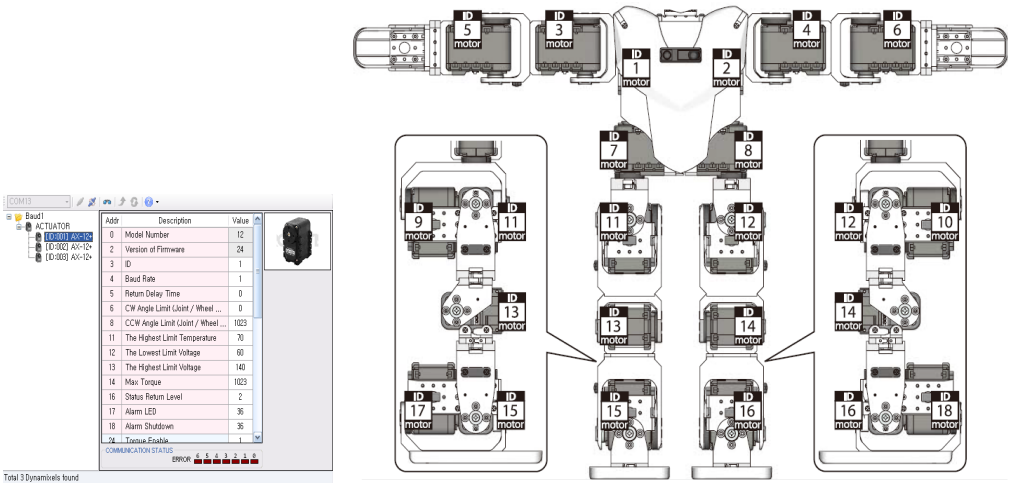
Figura 3.5: Detalle de la Electrónica

Clank cuenta con 20 grados de libertad, todos servoactuados. Los servomotores utilizados han sido los Dynamixel AX-12 para los 18 g.d.l que conforman el cuerpo y el modelo XL-320 para los servomotores que controlan el movimiento de la cabeza. Estos primeros motores funcionan a 12 voltios. Sn emabrgo, los dos motores tipo XL funcionan a un voltaje menor, por lo que se ha tenido que intercalar un reductor de tensión en el bus para llegar con la tensión correcta a éstos, que es de 7.5 V.

La utilización de los motores de Dynamixel representan una gran ventaja, puesto que, por un lado, están pensados para acoplarse a piezas o componentes de manera sencilla ya que traen distintos taladros, y, por otro, que incorporan un microprocesador digital que se puede comunicar mediante el bus TTL, esto, además persmite conectar todos los motores en serie, reduciendo de manera siginifativa el cableado a utilizar.

En la puesta a punto de los servomotores se ha utilizado la interfaz que proporcio-

na Dynamixel (Dynamixel Wizard) para el sistema operativo Windows. Este software permite establecer distintos parámetros tales como las constantes del PID del motor, habilitar o deshabilitar el torque, gestionar las alarmas, etc. Además se ha tenido que cambiar el identificador de cada motor en el bus, puesto que, por defecto, todos los servomotores traen el identificador 1. De esta manera, se han numerado del 1 al 20 de acuerdo a la figura 3.6b (b).



(a) Dynamixel Wizard (b) Relación servomotor-articulación

Figura 3.6: Instalación y montaje de los servomotores Dynamixel

3.2. Entorno software

En esta sección se pretende tratar el entorno software del robot Clank, tratando toda su arquitectura de programación, desde las instrucciones de alto nivel hasta las órdenes de movimiento a los motores, pasando por el tratamiento de la información de sus sensores.

3.2.1. Sistema Operativo Robótico. ROS

Desde un comienzo se pretendía desarrollar una programación modular, flexible y descentralizada. Es por ello que se ha apostado por usar el *Robotic Operating System* [10] como entorno de trabajo, en adelante ROS.

ROS es un *middleware* que provee librerías y herramientas para ayudar a los desarro-

lladores de software a crear aplicaciones para robots. Provee abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por mensajes, administración de paquetes y más. ROS está bajo la licencia open source.

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.

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.

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.

En concreto, la versión de ROS utilizada ha sido Kinetic Kame, pero cabe decir que en el ordenador a bordo del robot (Raspberry Pi 3b+) el sistema operativo es Raspbian con una arquitectura ARM, con lo que se han tenido que compilar desde código fuente las librerías de ROS. Sin emabrgo, el ordenador portátil externo al robot el sistema operativo es Ubuntu 16.04 de escritorio (amd64), con lo que ROS está incluido en los repositorios oficiales y, por tanto, no hay necesidad de compilarle desde código, esta descentralización se explicará más adelante.

Los archivos ejecutables dentro de un proyecto en ROS pueden ser, hasta el momento, de dos tipos: con extensión .cpp (C++) o .py (Python) y reciben el nombre de nodos. 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, etc [5].

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

Paradigmas de comunicación

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 3.7.

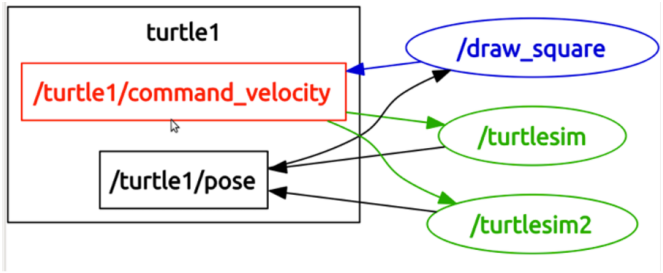


Figura 3.7: Modelo Editor Suscriptor

En la imagen 3.7 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 [5]..

Aunque el modelo editor-subscriptor es en el que se fundamentan los topics, también existen lo que se denominan *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 Clank.

Finalmente, existe otro tipo especial de comuicación, es la llamada librería de acciones. En cualquier sistema grande basado en ROS, hay casos en los que alguien desea

enviar una solicitud a un nodo para realizar alguna tarea y también recibir una respuesta a la solicitud. Esto se puede lograr actualmente a través de servicios ROS.

Sin embargo, en algunos casos, si el servicio tarda mucho tiempo en ejecutarse, es posible que el usuario desee la posibilidad de cancelar la solicitud durante la ejecución u obtener comentarios periódicos sobre el progreso de la solicitud. El paquete *actionlib* proporciona herramientas para crear servidores que ejecutan objetivos de ejecución prolongada que se pueden anular. También proporciona una interfaz de cliente para enviar solicitudes al servidor.

ActionClient y ActionServer se comunican a través de un "Protocolo de acción de ROS", que se basa en los mensajes de ROS. Luego, el cliente y el servidor proporcionan una API simple para que los usuarios soliciten objetivos (en el lado del cliente) o ejecuten objetivos (en el lado del servidor).

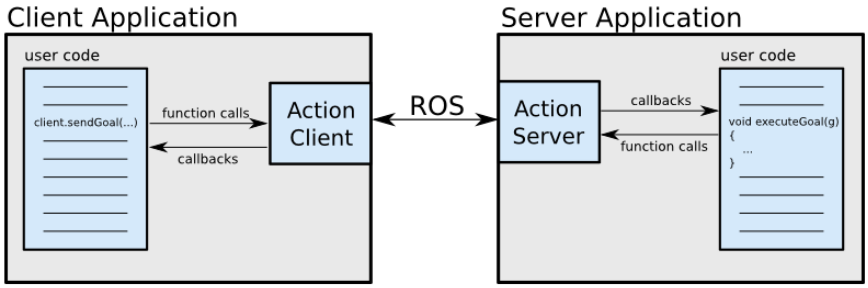


Figura 3.8: Acción

La especificación de una acción es: objetivo, realimentación y resultado. Para que el cliente y el servidor se comuniquen, necesitamos definir algunos mensajes sobre los cuales se comunican. Esto es con una especificación de acción. Esto define los mensajes de objetivo, comentarios y resultados con los que se comunican los clientes y los servidores:

Objetivo o *Goal*

Para realizar tareas utilizando acciones, introducimos la noción de un objetivo que un ActionClient puede enviar a un ActionServer. En el caso de mover una base móvil, el objetivo sería un mensaje *PoseStamped* que contenga información acerca de dónde debe moverse el robot en el mundo. Para controlar la inclinación del escáner láser, el objetivo

contendría los parámetros de escaneo (ángulo mínimo, ángulo máximo, velocidad, etc.).

Realimentación

La realimentación proporciona a los implementadores de servidores una forma de informar a un ActionClient sobre el progreso incremental de un objetivo. En una base móvil, esta podría ser la postura actual del robot a lo largo del camino. Para controlar la inclinación del escáner láser, este podría ser el tiempo restante hasta que se complete el escaneo.

Resultado

Un resultado se envía desde ActionServer hacia el ActionClient, al completar el objetivo. Esto es diferente a la realimentación, ya que se envía exactamente una vez. Esto es extremadamente útil cuando el propósito de la acción es proporcionar algún tipo de información. En el ejemplo de una base móvil, el resultado no es muy importante, pero podría contener la posición final del robot. Para controlar la inclinación del escáner láser, el resultado puede contener una nube de puntos generada a partir del escaneo solicitado. [11]

Herramientas para el desarrollo

- 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 se publican en tres dimensiones.

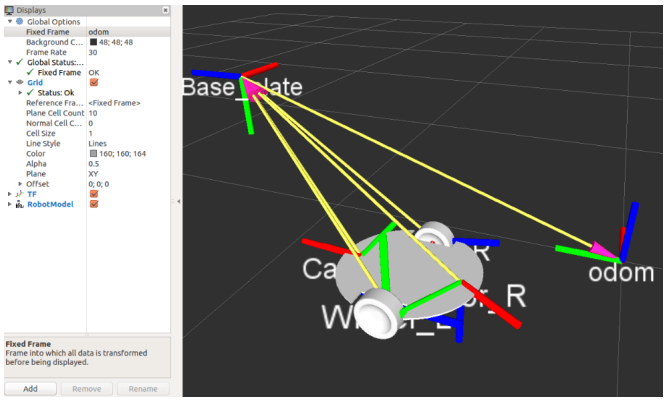


Figura 3.9: Rviz

- 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.

- Sistemas de referencia (*Transformed Frames*, o tf): tf es un paquete que permite realizar un seguimiento de múltiples sistemas de referencia a lo largo del tiempo. tf mantiene la relación entre los sistemas de coordenadas en una estructura de árbol almacenada en un búfer en el tiempo, y que permite al usuario transformar puntos, vectores, etc. entre dos sistemas de coordenadas en cualquier punto deseado en el tiempo. Se pueden visualizar en Rviz, como se muestra en la figura 3.9.

- Gazebo: se trata de un simulador en el que se pueden modelar distintos tipos de contactos mecánicos y sensores, por ejemplo, una unión prismática entre dos articulaciones de un brazo robot. De igual manera, se puede simular la fricción de una rueda con el suelo, para producir movimiento o un LIDAR escáner láser. Gazebo tiene varios plugins compatibles que lo hacen compatible con ROS.

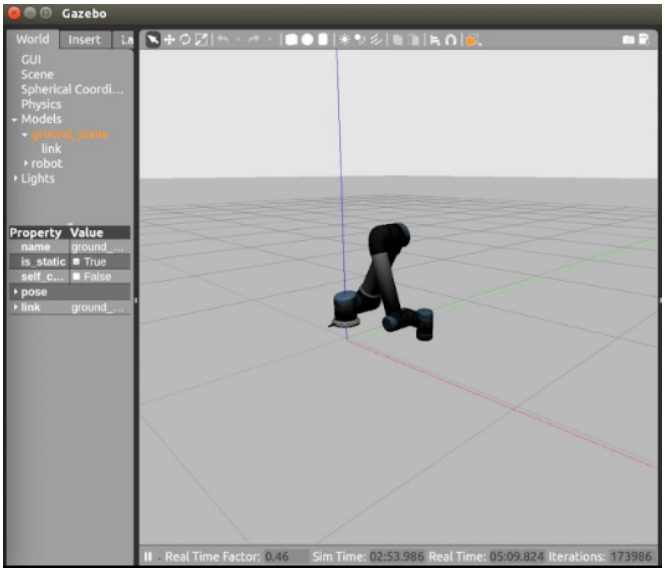


Figura 3.10: Gazebo

3.2.2. ROS en los sistemas en red

Como se ha comentado en la primera sección del capítulo, una de las grandes ventajas de ROS es su modularidad. Al final, los nodos se comunican entre sí usando puertos. Esto no es totalmente cierto en todos los casos, puesto que se existen lo que se laman «nodelets» que permiten el intercambio de mensajes mediante memoria compartida. Dichos nodelets se utilizan principalmente cuando se tienen que intercambiar mensajes muy grandes y pesados, como es el caso de imágenes o nubes de puntos.

Dejando al margen estos casos particulares, ROS permite la conexión entre equipos en red de manera sencilla y rápida. Basta con que los nodos exporten su IP y se especifique la dirección y puerto de la máquina que contiene el nodo maestro.

En Clank se aprovecha esta funcionalidad. Por un lado, en el ordenador de a bordo (la Raspberry Pi) de a bordo se lanzarán aquellos nodos correspondientes a la interfaz hardware, que obtienen datos de sensores y envían a actuadores, de forma resumida. Mientras que, por otro lado, en un ordenador portátil externo se lanzarán el maestro junto con los nodos más pesados correspondientes a la panificación de movimientos y cálculos complejos.

3.2.3. Estándar ROS Industrial

ROS-Industrial es un programa con licencia abierta BSD que contiene bibliotecas, herramientas y controladores que extienden las capacidades de ROS para hardware industrial. Es apoyado y guiado por el Consorcio ROS-Industrial. Los objetivos de ROS-Industrial son:

- Crear una comunidad apoyada por investigadores y profesionales de robótica industrial.
- Proporcionar una ubicación única para aplicaciones ROS relacionadas con la industria.
- Desarrollar software robusto y confiable que satisfaga las necesidades de las aplicaciones industriales.
- Combinar las fortalezas relativas de ROS con las tecnologías industriales exis-

tentes (es decir, la combinación de la funcionalidad de alto nivel de ROS con la confiabilidad y seguridad de bajo nivel de los controladores de robots industriales).

- Crear interfaces estándar para estimular el desarrollo de software ïndependiente del hardware"(utilizando mensajes ROS estandarizados).
- Proporcionar un camino fácil para aplicar la investigación de vanguardia en aplicaciones industriales, utilizando una arquitectura ROS común.
- Proporcionar API sencillas, fáciles de usar y bien documentadas.

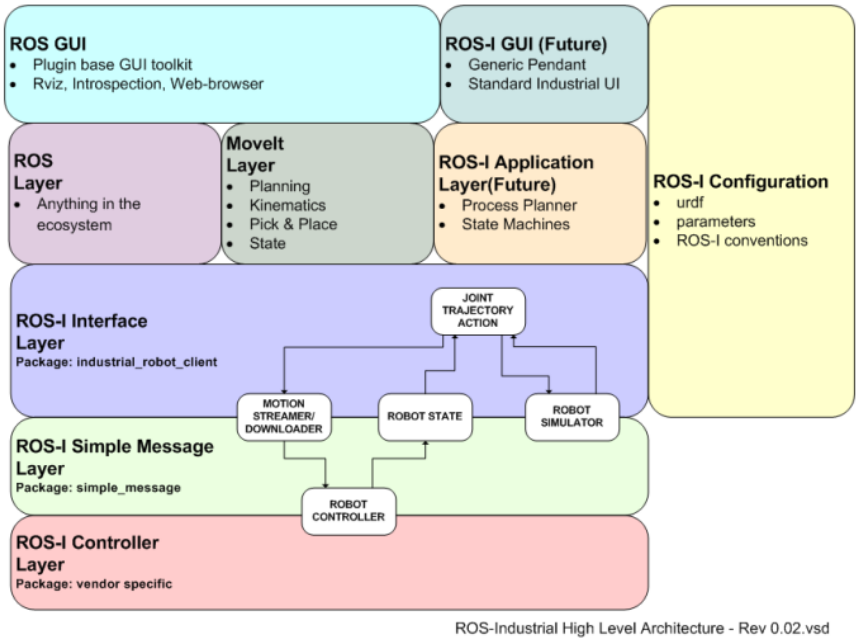


Figura 3.11: ROS Industrial

Paradigma ROS Control

El conjunto de paquetes denominados ROS control [12] proporciona la capacidad de implementar y administrar controladores de robot con un enfoque tanto en el rendimiento en tiempo real como en el uso compartido de controladores de una manera independiente del robot. La motivación principal para un entorno de control de robot es la falta de una capa de comunicación *realtimesafe* en ROS. Además, el marco implementa soluciones para el ciclo de vida del controlador y la gestión de recursos de hardware, así como las abstracciones en las interfaces de hardware con suposiciones

mínimas sobre el hardware o el sistema operativo. El diseño modular de ros_control lo hace ideal tanto para investigación como para uso industrial.

El eje fundamental de las librerías es la capa de abstracción de hardware, que sirve como un puente hacia diferentes robots simulados y reales. Esta abstracción es proporcionada por la clase hardware_interface::RobotHW por tanto, las implementaciones específicas de robots tienen que heredar de esta clase. Las instancias de esta clase modelan recursos de hardware proporcionados por el robot tales como actuadores eléctricos e hidráulicos y/o sensores de bajo nivel tales como codificadores y sensores de fuerza / par.

Existe la posibilidad de componer instancias de RobotHW ya implementadas que es ideal para construir sistemas de control para robots donde las piezas provienen de diferentes proveedores, y cada uno suministra su propia instancia específica de RobotHW.

El *controller_manager* es responsable de administrar el ciclo de vida de los controladores, y recursos de hardware a través de las interfaces y manejo de conflictos de recursos entre controladores. El ciclo de vida de los controladores no es estático sino que puede ser consultado y modificado en tiempo de ejecución a través de los servicios ROS estándar proporcionados por el controller_manager. Tales servicios permiten iniciar, detener y configurar controladores en tiempo de ejecución. Además, ros_control lleva consigo bibliotecas de software que abordan la comunicación, las transmisiones y los límites conjuntos de ROS en tiempo real. La biblioteca *realtime_tools* agrega clases de mucha utilidad para manejar las comunicaciones de ROS de manera segura en tiempo real.

El paquete *transmission_interface* proporciona clases que implementan conversiones entre el espacio de articulaciones y el actuador, tales como: reductor simple, enlace de cuatro barras y transmisiones diferenciales. La definición declarativa de las transmisiones se admite directamente con la descripción de cinemática y dinámica en el archivo de formato de descripción de robot universal (URDF) (Willow Garage 2009) del robot (se ampliará en el siguiente capítulo). El paquete *joint_limits_interface* contiene estructuras de datos para representar límites conjuntos y métodos para establecerlos a través de archivos URDF o yaml. *control_toolbox* por su parte ofrece componentes

útiles al escribir controladores: un controlador PID, filtros, Generadores sinusoidales y de ruido.

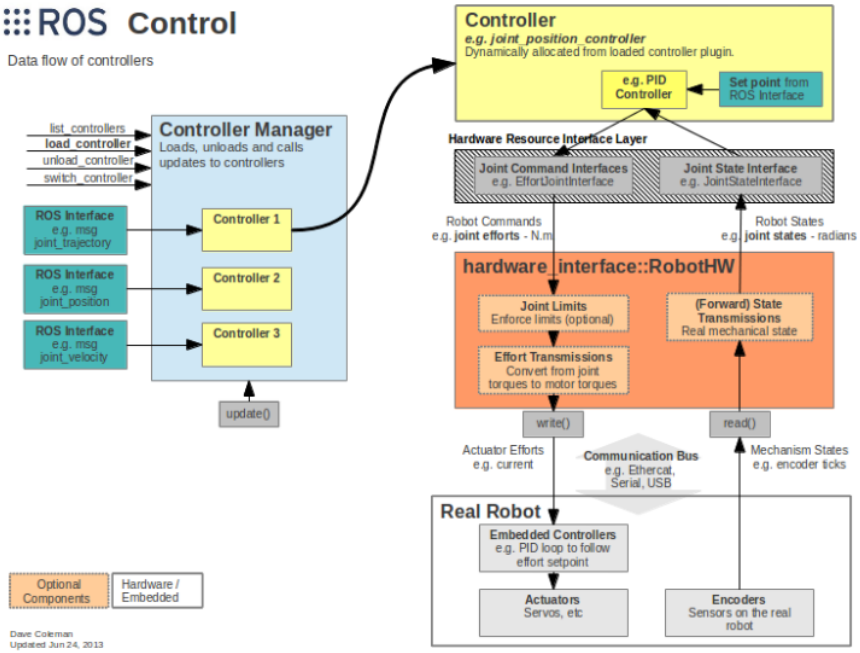


Figura 3.12: ROS Control

Capítulo 4

PLANIFICADOR DE TRAYECTORIAS. MOVEIT

En el siguiente capítulo se centra en el desarrollo del planificador de trayectorias. La decision de aplicar el estándar ROS Industrial trae consigo el uso y aprendizaje de distintas herramientas, desde la descripción hardware del robot hasta el planificador Moveit.

4.1. Descripción del robot. URDF

El *Universal Robot Description Format o URDF* es la manera adoptada en ROS de describir el robot mediante la definición de sus articulaciones en un formato de texto tipo XML. Esta descripción consta de distintas etiquetas con estructura de árbol (figura 4.1), que se describen a continuación:

- Etiqueta *Link*: son cada una de las «piezas» que conforman el robot, van ligadas a un sistema de referencia (tf, como se explicaba en el capítulo anterior) y como parámetro se pasa un nombre. Además tiene distintas etiquetas «hijas» ligadas, como es el tag «visual» que incluye el elemento visual de la pieza, y que, a su vez, contiene distintas etiquetas como son «geometry» que define una forma simple como un prisma, o una esfera, con sus dimensiones, la etiqueta «mesh» donde se explicita la ruta a un fichero con extensión STL o DAE (de diseño CAD, 3D) o

la etiqueta «origin» en la que se puede definir el origen del sistema de referencia. Un ejemplo de definición sería el siguiente:

<link name="gripper»

<visual>

<geometry>

<mesh filename="package://pkg/m.dae/>

</geometry>

</visual>

</link>

- Etiqueta *Joint*: esta etiqueta define la articulación de dos piezas o *links*. De esta forma, la articulación puede ser prismática, de rotación limitada, de rotación continua, o fija. Se espera la definición de un nombre «name» y de un tipo «type» (fixed, continuous, revolute...) como parámetros. Además, tiene diferentes «tags» como hijos:

  - *parent*: especifica el nombre del «Link» padre.
  - *child*: especifica el nombre del «Link» hijo.
  - *origin*: identifica en dos ternas de tres valores, la posicion y orientación del sistema de referencia hijo respecto del padre.
  - *axis*: en caso de ser una articulación no fija prismática o de revolución, se especifica el eje en el que tiene lugar el grado de libertad, en caso de ser de revolución, el eje de giro, y en caso de articulación prismática el eje de desplazamiento.

A continuación se muestra un ejemplo de definición de un «joint» entre el antebrazo «forearm» y la pinza «gripper» de un robot manipulador.

<joint name="joint1"type=revolute»

<parent link="forearm/>

<child link="gripper/>

<origin xyz="0.5 0 0rpy="0 0 -1.57/>

<axis xyz="0 0 1/>

</joint>

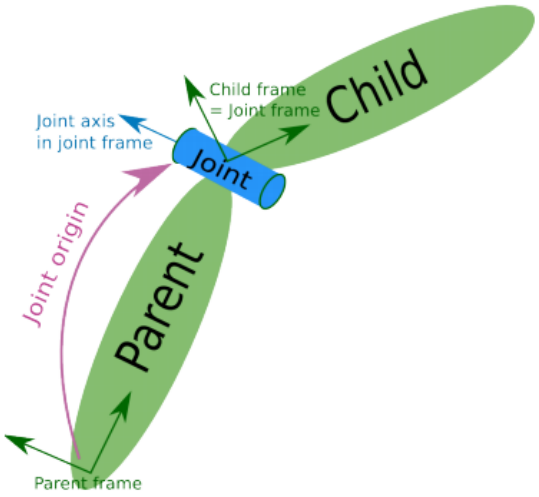


Figura 4.1: Modelo en URDF

4.2. Xacro

En la sección anterior se ha descrito cómo se modela un robot de acuerdo al modelo URDF. A menudo, los robots suelen tener un número elevado de articulaciones, por ejemplo, un brazo tipo antropomórfico de 6 ejes de libertad, o un robot móvil en la que del sistema de referencia base cuelgan numerosos conjuntos de ruedas, en tracción y dirección.

Es por ello que surje la necesidad de ampliar las capacidades del modelo URDF pudiendo llamar desde un fichero principal del robot a otros ficheros en los que se crean las denominadas «macros», que se pueden interpretar como *clases* o *estructuras* que agrupan partes reducidas de la descripción del robot. A este nuevo tipo de fichero se le conoce como XACRO.

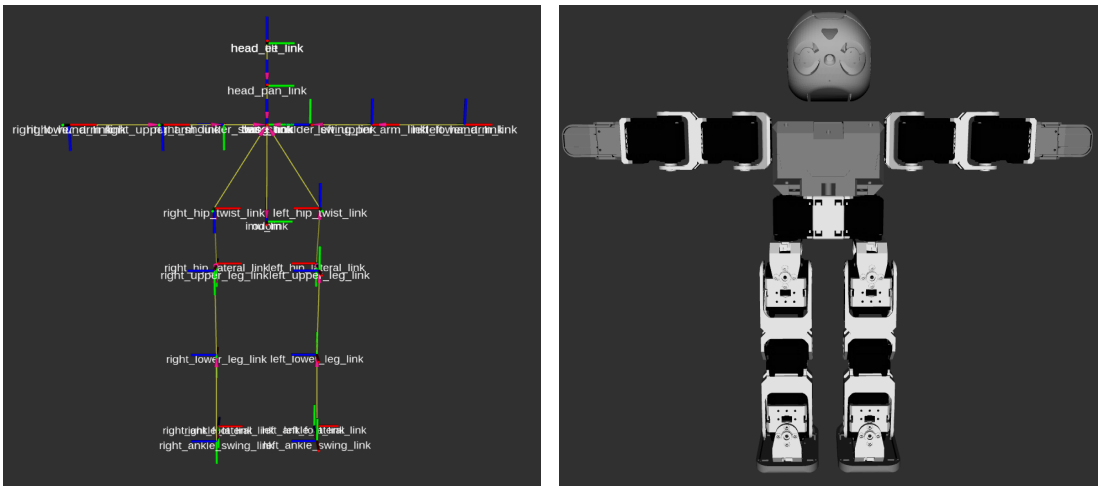
De esta forma se pueden agrupar en una macro, por ejemplo, el conjunto formado por el grupo de dirección-tracción. Desde el fichero fuente de la plataforma base, se

pueden declarar tantas macro de tipo «motor-dirección» como disponga el vehículo.

Además, una macro puede recibir argumentos cuando esta es invocada desde otro fichero y dentro se pueden definir sentencias tipo «if else». Esto posibilita, siguiendo con el ejemplo del robot móvil, que se puedan definir conjuntos «motor-dirección» a izquierda y derecha reutilizando la misma macro, pero cambiando el argumento «prefijo», a la hora de invocarlas.

Al modelo de un robot en XACRO o URDF se le pueden añadir plugins que contienen unas etiquetas especiales si se desea simular el robot en Gazebo (plugin de ros control). Dichas etiquetas o «tags» contienen información acerca del tipo de transmisión mecánica de cada articulación, si es controlada por posición, velocidad o torque, además de la definición de coeficientes de rozamiento etc... Esto posibilita al usuario analizar de manera previa el movimiento del robot en simulación.

Clank está modelado en Xacro (figura 4.2b), siendo el *link* llamado «odom» el t.f raíz, u origen del árbol. De él cuelgan el torso, la cintura



(a) TF de Clank (b) Clank en Rviz

Figura 4.2: Descripción de Clank

4.3. Controladores e Interfaz hardware

La aplicación del concepto de ROS control al robot Clank, implica abstracción del hardware. En este sentido, los controladores se desacoplan del robot, esto quiere decir

que, de cara a los controladores y toda la capa de alto nivel de planificación les es «transparente» que el usuario esté simulando el robot en Gazebo o estar usando el robot real.

Esto es posible debido a que los paquetes de interfaz hardware de «ros control» extienden las capacidades y la información que se almacena en los sistemas de referencia, de forma que en ellos ahora residen los «próximos comandos a ejecutar» (a través de la activación de «callbacks») y, por otro lado, la información que se obtiene de los sensores reales o simulación, que actualiza la transformada entre ellos.

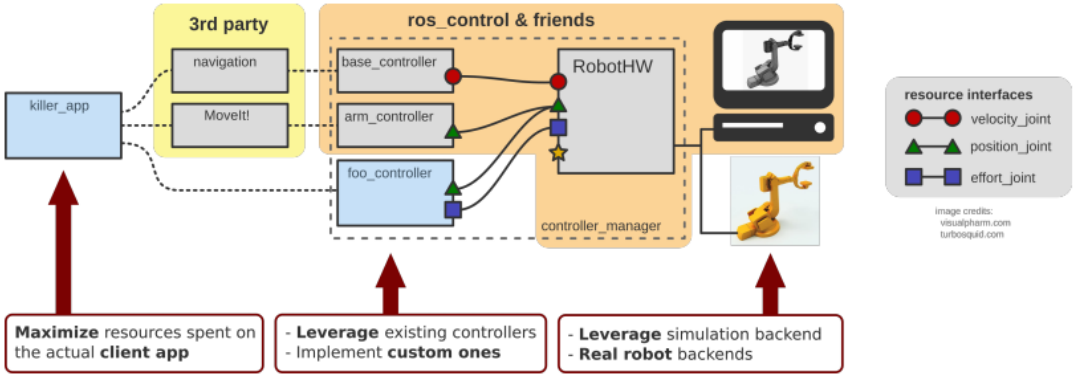


Figura 4.3: Vista Genera de la interfaz Hardware

Los controladores de Ros control tienen la capacidad de ejecutarse en sistemas de tiempo real. Sin embargo es habitual encontrarse cierta confusión entre la comunidad en este aspecto. Ha de aclararse que para que dichos controladores sean, en efecto, de tiempo real, el sistema operativo en el que se ejecutan tiene que ser de tiempo real, así como el bus y las comunicaciones entre la CPU y el robot. Por defecto el kernel de Linux no está preparado para ejecutarse en tiempo real, auque se puede añadir un parche para que lo sea. La principal diferencia se halla en la habilitación o no de una directiva PREEMPT_RT que hace que el S.O pueda desalojar siempre una tarea, eso garantiza el una tasa fija de tiempo de ejecución de ciclo, es decir, tiempo real.

Por otro lado es interesante comentar que aunque el sistema operativo no sea de tiempo real, el comportamiento puede ser muy cercano, aquí se debe atender al concepto de *granularidad*. De tal manera que si la programación de una tarea compleja se divide en muchas tareas pero sencillas y ágiles de ejecutar se hablará de granularidad fina,

y en caso contrario, de granularidad gruesa. Aquellos procesos con granularidad fina tienen un comportamiento comparable a uno que se ejecutara sobre un S.O de tiempo real. Si se desea profundizar más en el concepto de tiempo real, referirse al nuevo ROS2[1] actualmente en desarrollo, en el que este concepto de tiempo real toma mucha importancia.

El conjunto de estos controladores se lista en un archivo tipo .yaml y tienen la estructura siguiente, por ejemplo el controlador de la pierna izquierda:
left_leg_controller:
type: position_controllers/JointTrajectoryController
joints:
- left_hip_twist_joint
- left_hip_lateral_joint
- left_hip_swing_joint
- left_knee_joint
- left_ankle_swing_joint
- left_ankle_lateral_joint

La especificación del tipo significa que la interfaz hardware que va a usar es de tipo posición y que el comando que recive el contrador también es de tipo posición. El lazo de control de posición ocurre en el mismo servomotor, que es donde se ubica el PID.

4.4. MoveIt

Los robots están encontrando cada vez más aplicaciones en dominios donde tienen que trabajar de cerca o en proximidad a los humanos. Las aplicaciones robóticas industriales están comenzando a examinar la posibilidad de robots y humanos como compañeros de trabajo, compartiendo tareas y espacio de trabajo. Los coches que operan en calles concurridas y autopistas tienen que compartir espacio con peatones y ciclistas además de otros vehículos. Los robots domésticos, en particular los sistemas de manipulación móviles, se enfrentarán con ambientes desordenados y desestructurados donde

[1] https://index.ros.org/doc/ros2/

los obstáculos existen en cada esquina, y las personas moviéndose continuamente dentro y fuera del espacio de trabajo de los robots.

Los robots que trabajan en entornos humanos tienen claramente que ser conscientes de su entorno y deben intentar activamente evitar colisiones con humanos y otros obstáculos MoveIt es un conjunto de paquetes de software integrados con ROS y diseñado específicamente para proporcionar tales capacidades, especialmente para manipulación móvil. MoveIt permitirá robots para construir una representación de su entorno utilizando datos fusionados de Los sensores tridimensionales (3-D) y otros, generan planes de movimiento que mueven al robot de manera efectiva y segura en el medio ambiente, y ejecutar el planes de movimiento mientras monitorea constantemente el entorno para detectar cambios.

4.4.1. Orígenes

MoveIt es una evolución del software arm_navigation en ROS. Los paquetes de arm_navigation fueron diseñados para planificación del movimiento, generación de trayectorias y monitoreo del entorno para brazos robotizados (ej, PR2) a través de los datos de sensores láser y estéreo. Todo esto se fusionó en el robot PR2 para generar un modelo coherente del entorno. El modelo de entorno puede manejar oclusiones de partes del cuerpo del robot y también ruido significativo en datos del sensor 3-D. El ambiente era representado como una mezcla de dos formatos:

1) Una rejilla que representa la mayor parte de los obstáculos en el medio ambiente y

2) Primitivas geométricas y modelos de malla para representar objetos que habían sido reconocidos y registrados en el entorno por rutinas de detección de objetos.

El modelo de entorno fue construido sobre el paquete Octomap, una representación probabilística basada en octbase, permitiendo una representación eficiente de grandes escenas desordenadas. El entorno modelo sirvió como la entrada principal para el uso de planificadores de movimiento rápido que podían generar rápidamente planes de movimiento sin colisiones en entornos bastante complejos. Una Interfaz genérica en ROS para la planificación geométrica ha permitido la integración de múltiples planificadores

de movimiento, incluyendo planificadores aleatorios de Open Motion Planning Library (OMPL), planificadores basados en búsquedas (SBPL) [3], bibliotecas de optimización de trayectoria que incluyen CHOMP, y optimización de la trayectoria estocástica para la planificación del movimiento (STOMP).

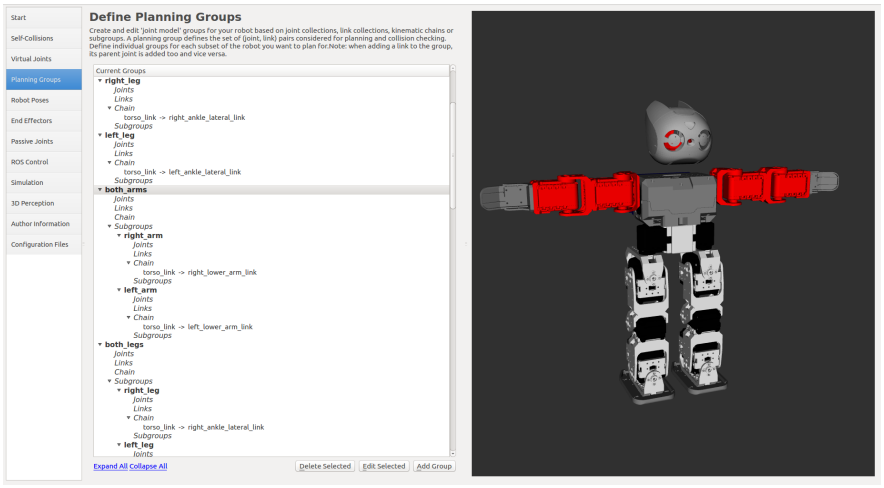
Algunos de los planificadores de movimiento eran capaces de lidiar con restricciones geométricas, por ejemplo, una restricción de orientación especificando que un vaso de agua debe mantenerse erguido. Los suavizadores de ruta y los generadores de trayectoria se utilizaron para parametrizar las rutas planificadas y para que las trayectorias pudiesen ser ejecutadas en el robot. Para obtener soluciones al problema cinemático inverso se utilizó un componente llamado move_arm que formó el interfaz principal para este conjunto de capacidades, exponiendo una interfaz que permitió que los usuarios especificaran conjuntos de puntos en un espacio cartesiano.

4.4.2. Portabilidad

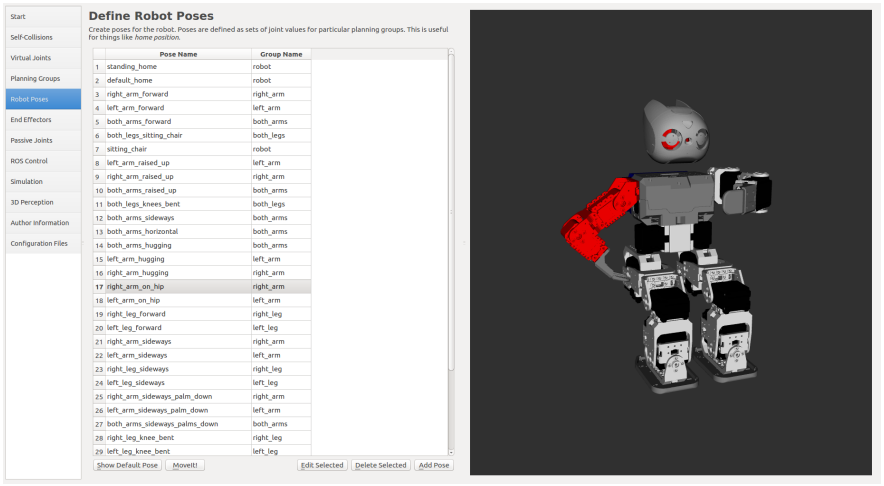
El software de arm_navigation fue diseñado principalmente para el robot PR2. Sin embargo ha sido utilizado en varios sistemas robóticos. En MoveIt, se ha hecho un esfuerzo adicional para extraer toda la información necesaria en el existente URDF y recién definido Formato de descripción de robot semántico (SRDF) para evitar dependencias de robots específicos. Además, se incorporan nuevas herramientas como el Asistente de navegación del brazo (que analizaremos en la siguiente sección), diseñado específicamente para permitir la configuración del software arm_navigation para otros manipuladores robóticos y sistemas que están siendo portados a MoveIt.

4.4.3. Asistente para la configuración de Moveit

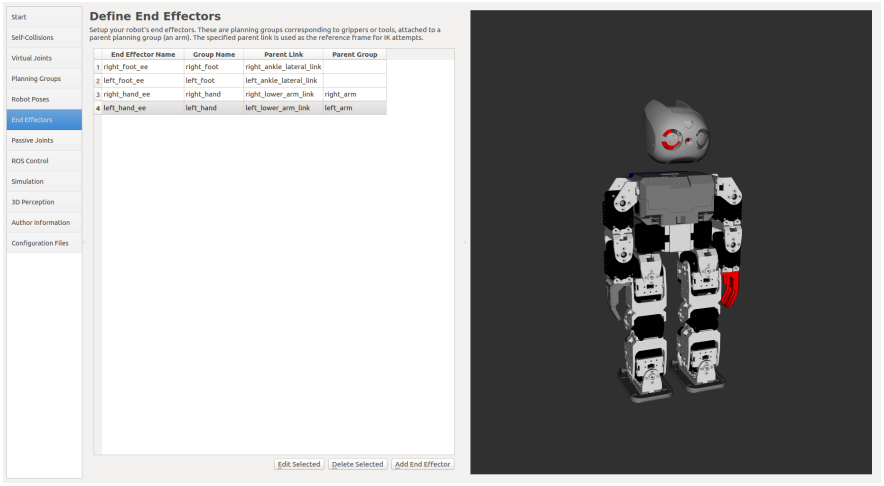
El asistente para la configuración de Moveit utiliza una combinación de URDF y SRDF para autoconfigurar el software de arm_navigation en un robot. Una interfaz gráfica interactiva de usuario permite a estos especificar el plan de movimiento para nuevos robots, con una cantidad mínima de implementación de interfaz requerida por el usuario en el lado del robot. Esto permite que no sean necesarios expertos en planificación de movimiento para configurar fácilmente la planificación de movimiento y los componentes asociados en ROS para sus propios robots.



(a) Grupos de planificación en Moveit Setup Assistant



(b) Posiciones prefijadas en Moveit Setup Assistant



(c) Efectores en Moveit Setup Assistant

Figura 4.4: Clank en Moveit Setup Assistant

Dentro del asistente se pueden definir cadenas cinemáticas correspondientes a conjuntos de articulaciones, de esta manera, en Clank, se tienen varias cadenas cinemáticas principales, cada brazo (izquierdo y derecho), ambos brazos, cada pierza (izquierda y derecha), ambas piernas y la cabeza. Para cada grupo de planificación individual se pueden, además, configurar actuadores de final de cadena como son pinzas, por ejemplo (grippers) y un plugin para el cálculo de la cinemática inversa (habitualmente KDL).

Lo que este asistente de configuración finalmente genera es un paquete de distintos ficheros correlaccionados entre ellos que definen y parametrizan el planificador de acuerdo a la configuración establecida en el asistente.

Este planificador configurado por MoveIt en el fondo se trata de un cliente de acción para el servidor de acción que generan los controladores de ros_control, en concreto, de los del tipo joint_trajectory_controller. Estos controladores incluye todas las articulaciones del robot Clank, por lo que puede planificar trayectorias para todos los conjuntos de articulaciones: cabeza, brazo izquierdo, brazo derecho, pierna izquierda y pierna derecha.

4.5. Planificador de alto nivel. Move Group

En MoveIt, la interfaz de usuario principal es a través de la clase MoveGroup. Proporciona una funcionalidad de alto nivel para la mayoría de las operaciones que un usuario puede querer realizar, específicamente establecer objetivos en el espacio de articulaciones o llegar a un determinado punto en el espacio, crear planes de movimiento para mover el robot.

Además, dentro de la interfaz se pueden visualizar en Rviz los planes procesados, se puede planificar para más de un brazo de manera simultánea, y agregar objetos al entorno y acoplar/separar objetos del robot (en la escena).

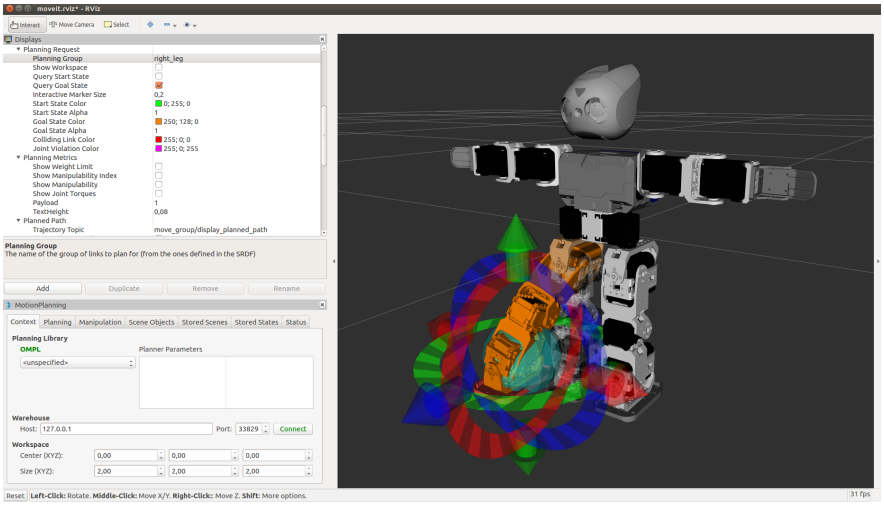


Figura 4.5: Clank Move Group en Rviz

Capítulo 5

CONTROL DE ESTABILIDAD

A partir de la capacidad de saber la orientación de Clank y disponiendo de una interfaz hardware completa se ha desarrollado un sistema de estabilidad de balanceo, que se pretende profundizar a lo largo de este capítulo.

5.1. IMU. Acelerómetro, giróscopo y magnetómetro

Como se introducía en el tercer capítulo, Clank cuenta con un sensor inercial, que consta de tres elementos: un acelerómetro, un giróscopo y un magnetómetro. El sensor está ubicado en la zona de la pelvis del robot (aproximadamente en su centro de masa) y se encuentra conectado mediante i2c a la placa OpenCM. En el microcontrolador se encuentra programada una calibración para el sensor y, a su vez, se comunica con la Raspberry Pi via serie.

Ya se han introducido a lo largo del capítulo cuarto los sistemas de referencia que definen el robot. El sistema de referencia padre es el de la odometría (cálculo de cúanto se mueve el robot a partir del cómputo del giro de los servomotores). En el REP-105 de ROS[1] se define el nombramiento y estructura de los sistemas de referencia para el uso de os robots móviles. En concreto, dentro del sistema odometría se incluyen todos los sensores de flujo de datos continuo y relativos al momento de arranque. Por otro lado, si se disponen de sistemas de localización absolutos tanto en interiores (láser, mapas,

[1] https://www.ros.org/reps/rep-0105.html

reflectores, códigos de barras, etc) o exteriores (gps) entonces el flujo de dichos datos computaría la transformada entre el sistema referencia mapa y odometría, para que la posición entre mapa y robot sea la que devuelve dicha localización.

El sistema de referencia hijo de la odometría es el llamado «imu_link». El primer paso, por tanto, es el de obtener y publicar la transformada entre estos dos sistemas de referencia (la orientación, «roll, pitch y yaw», cabeceo, alabeo y guiñada) con los datos que proporciona el sensor inercial.

5.2. Calibrado y Filtrado de la señal

En la placa OpenCM tiene lugar la calibración del sensor de inercia. Para cada elemento que compone el sensor se toma su sensibilidad, de esta forma:

- El acelerómetro tiene una resolución digital de 16 bits, una rango de +-2g una sensibilidad de 0.061 mg/dígito. En general, la aceleración en estos sensores se mide en g. Un g vale aproximadamente 9,81 (el valor de la gravedad). Por tanto la ecuación para convetir los datos a g en cada eje, se aplicará:

$$accel = accel_{raw} * 0{,}061/1000{,}0 \tag{5.1}$$

- El magnetómetro tiene una resolución de 16 bits, un rango de +-2 gauss y una sensibilidad de 0.080 mgauss/dígito.

- El giróscopo tiene una resolución de 16 bit, un rango de +- 245 grados por segundoy una sensibilidad de 8.75 mgps/dígito, por tanto su ecuación será, en radianes por segundo:

$$gyro = gyro_{raw} * 0{,}00875 * PI/180 \tag{5.2}$$

  En este último caso se hace necesario aplicar un offset al valor de salida para cada eje, de manera que la media de los valores, aunque con dispersión, sea cero, para

cuando el robot esté estático. Estos valores son de 0.0488, 0.18387 y 0.0605 para, respectivamente, x, y, z.

Esto conforma la salida de los datos hacia el PC, donde tiene lugar el filtrado de la señal y la publicación de la transformada entre los sistemas odometría e imu.

Los datos del acelerómetro y magnetómetro se almacenan en variables internas a la clase, de forma que se actualizan con la llegada de un nuevo callback. Cuando llegan nuevas medidas del giróscopo, lo primero que tiene lugar es introducir estos datos en unos búffer de tamaño 20, y el valor tomado para ese instante de tiempo es el valor medio del búffer. Esta es una primera manera de filtrar posible ruido. A continuación, con cada recepción (callback) tiene lugar el siguiente algoritmo:

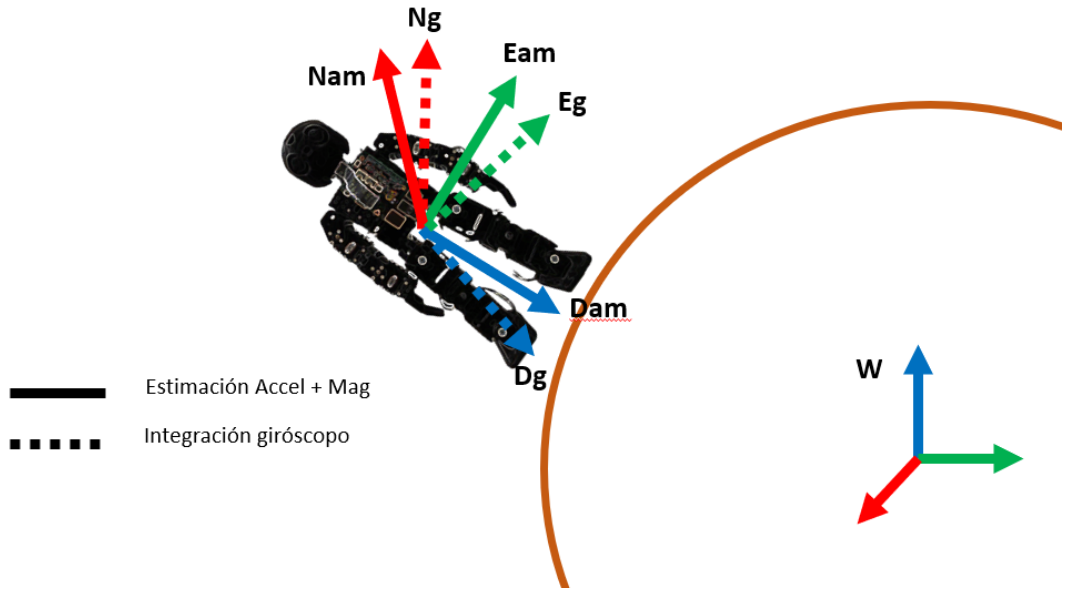
```
/* Llegada de un nuevo dato del giroscopo */

dt = tiempoAhora() - prevt

bufferx.push(gyro.x)
buffery.push(gyro.y)
bufferz.push(gyro.z)

velAngular.x = media(bufferx)
velAngular.y = media(buffery)
velAngular.z = media(bufferz)

filtroCoeff = constTiempo / (constTiempo + dt)

corregirOrientacion()
actualizarOrientacion()

prevt = tiempoAhora()
publicarTransformada()
```

En especial son interesantes los métodos «corregirOrientacion()» y «actualizarOrientacion()». En el primer método tiene lugar el algoritmo de corrección de la deriva que se produce en la integración del giróscopo, mientras que en el segundo es dónde se integran propiamente los datos del giróscopo:

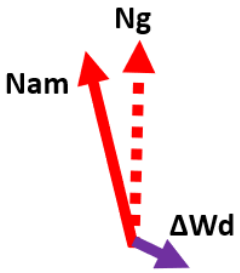
```
/* Metodo corregirOrientacion */

Si módulo(acceleracion) es aprox 1g:

  Vector3 Down = accel
  Vector3 East = Down x magnet
  Vector3 North = East x Down

  normalizar(Down)
  normalizar(East)
  normalizar(North)

  Matrix3x3 gyroAngRotMat = quaternioAMatriz(q)

  VectorCorrecion = filtroCoeff * Vector3(North x gyroAngRotMat.row0,
  East x gyroAngRotMat.row1, Down x gyroAngRotMat.row2)

  velAngular += VectorCorrecion

/* Metodo actualizarOrientacion */

Quaternio nuevoQuat = Quaternio(velAngular*dt)
q *= nuevoQuat
normalizar(q)
```

Como se ha dicho, los datos del giróscopo se van integrando en el tiempo en el método actualizarOrientacion(). Aquí cabe decir que en el álgebra de quaternios la

operación de multiplar es equivalente a la suma en el espacio angular normal, de tal manera que se van sumando los incrementos angulares.



(a) Distribución de los sistemas de referencia



(b) Factor de correción

Figura 5.1: Correción de orientación en IMU

La idea del procedimiento de correción es poder estimar cuál debería ser la orientación del robot a través del conocimiento del vector acceleración y de la dirección del campo magnético terrestre. Cuando el robot está estático (sin affección de fuerzas externas) entonces el módulo del vector acceleración tomará un valor cercano al de la gravedad (1g approx) y su vector apuntará hacia abajo (figura 5.1a Dg).

Por otro lado, se sabe que el magnetómetro del IMU está calibrado para señalar el este, por lo que se aprovecha dicho vector para señalar el este, y normalizarlo. El

norte se calcula como el producto vectorial de estos dos últimos vectores. Bien, pues la orientación de este sistema de referencia respecto de la Tierra otorga una estimación de cúal tendría que ser la orientación del robot en ese momento.

Por tanto, lo siguiente es «comparar» la orientación actual obtenida por integración del giróscopo con la estimada y formar un vector de correción. Dicha comparación se hace tomando el producto vectorial entre dos vectores (figura 5.1b), que hará que, cuanta más distinta sea la orientación actual de la estimada, más corrección se efectuará en ese eje. Aquí se incluye también un coeficiente de filtrado para amortiguar los cambios y que la corrección sea lo más suave posible.

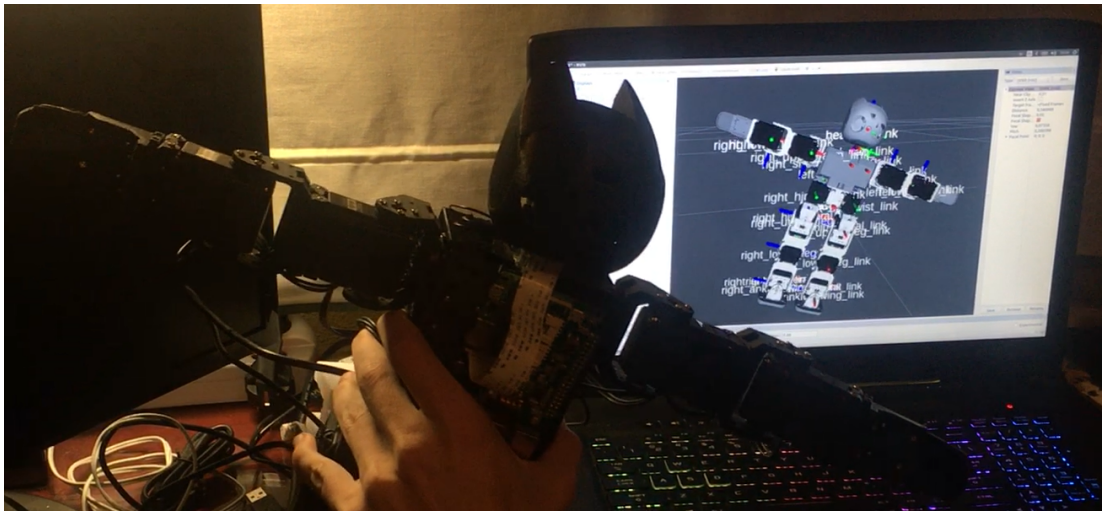


Figura 5.2: Rotación obtenida

5.3. Lazo de control de balanceo

La capacidad de poder medir la orientación de Clank en cada instante que se explicaba en la sección anterior y la posibilidad de actuar sobre los servomotores hace que sea posible tener control sobre la estabilidad del robot. En concreto, se ha probado la respuesta ante perturbaciones en el plano de cabeceo. El lazo de control que se propone se muestra en la figura 5.3.

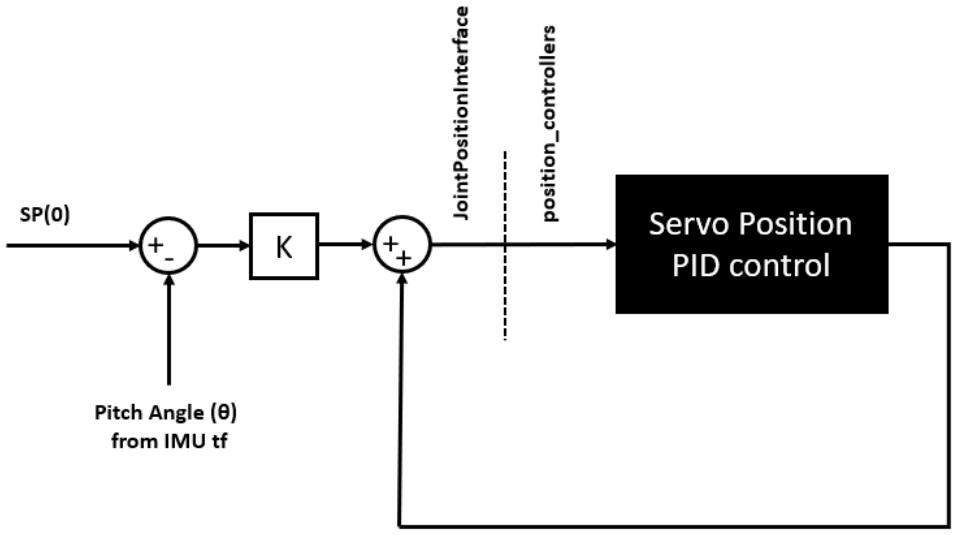


Figura 5.3: Lazo de Control de cabeceo

Cuando se somete al robot a un plano inclinado en el plano de cabeceo, del sensor de Inercia se puede obtener ese mismo desplazamiento angular respecto de la horizontal, es lo que en la figura se denota como *theta*. Con el control se pretende hacer que el robot se mantenga en la horizontal (utilizando los tobillos), con lo que se pretende hacer dicho desplazamiento igual a cero, que es la referencia (en la figura SP). Tras restar ambos valores, se multiplica el error por una ganancia (actualmente de valor 0.8) para entonces sumar o restar de la posición actual del servomotor este resultado.

Como se explicaba en el capítulo anterior, los controladores de los grupos de las piernas son de tipo «position_controllers/JointPositionInterface», esto es que los controladores reciben un comando de posición y su salida es un comando de tipo posición, por lo que, en realidad, en lo que conocemos como controladores de ros_control no está teniendo lugar ningún control PID.

Capítulo 6

SEGUIMIENTO FACIAL POR IMAGEN

Este capítulo comprende el desarrollo del sistema de seguimiento facial del robot Clank por imagen. Se estudiará el algoritmo que se aplica sobre la imagen obtenida de la cámara, el control efectuado y la salida de movimiento a los servomotores de la cabeza del robot mediante interfaz hardware.

6.1. Introducción

Cuando se trata de mejorar la interacción hombre-máquina, uno de los factores que influye de manera significativa es la expresividad y naturalidad del robot, esto es que se intente parecer lo más posible a cómo interacciona un humano. En este sentido, una regla social básica aceptada en una conversación entre dos personas es la de mirar a la persona con la que se está interactuando. Por tanto, el dotar al robot Clank de dicha capacidad es interesante por dicha justificación.

Según la fuente sensora se puede hacer seguimiento sobre una nube de puntos en 3D para, por ejemplo, obtener el esqueleto de la persona que se tiene delante, como se proponía con el robot Sacarino 2 en [5] y así identificar la posición de la cabeza y poder efectuar el seguimiento.

En concreto, Clank dispone de una cámara compatible con su Raspberry Pi dispuesta en el medio de la cabeza, entre los ojos (figura 6.1) conectada a ésta mediante el puerto CSI, por lo que el seguimiento se realizará a través del tratamiento de la imagen.

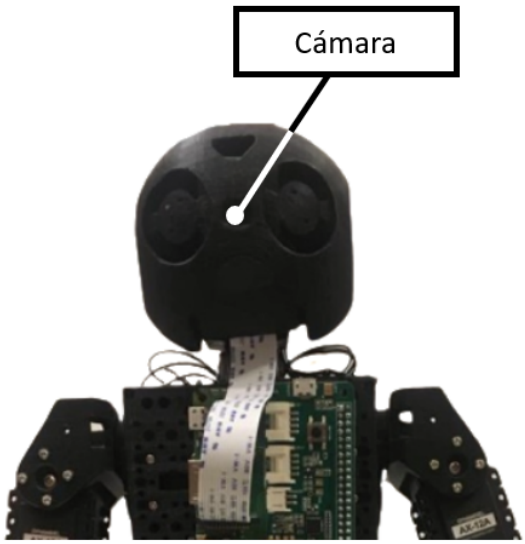


Figura 6.1: camara en Clank

Por otro lado, la cabeza de Clank tiene dos grados de libertad, en los planos de cabeceo y alabeo, que gobiernan dos servomotores, es en ambos planos en los que el robot será capaz de centrar su mirada con el usuario y que coinciden con los ejes coordenados del sistema de referencia imagen.

6.1.1. OpenCv

OpenCv [1] (Open Source Computer Vision) es una librería de visión por computador de código abierto. La librería está escrita en los lenguajes C y C++ y es compatible con Linux, Windows y Mac OS X. Cuenta con un desarrollo activo en interfaces para Python, Ruby, Matlab y otros lenguajes.

OpenCV ha sido diseñado para ser eficiente en cuanto a gasto de recursos computacionales y con un enfoque hacia las aplicaciones de tiempo real, además está escrito y optimizado en C y puede tomar ventaja de los procesadores con múltiples núcleos.

La librería OpenCV contiene aproximadamente 500 funciones que abarcan muchas

[1] https://opencv.org

áreas de la visión artificial, incluyendo inspección de productos de fábricas, escaneo médico, seguridad, interfaces de usuario, calibración de cámaras, robótica....etc, porque la visión por computador y el aprendizaje automática van de la mano.

OpenCV también tiene una completa librería de uso general de aprendizaje automático (MLL o Machine Learning Library), la cual es muy útil para cualquier problema de aprendizaje automático. Esta sublibrería está especializada en el reconocimiento estadístico de patrones y clustering.

Módulos de OpenCV:

OpenCV tiene una estructura modular. Los módulos principales de OpenCV se listan a continuación :

- Core: Este es el módulo básico de OpenCV. Incluye las estructuras de datos básicas y las funciones básicas de procesamiento de imágenes. Este módulo también es usado por otros módulos como highgui.
- Highhui: Este módulo provee interfaz de usuario, códecs de imagen y vídeo y capacidad para capturar imágenes y vídeo, además de otras capacidades como la de capturar eventos del ratón... etc. Si necesitas capacidades de UI (User Interface) más avanzadas debes usar frameworks tales como Qt, WinFormsetc.
- Imgproc: Este módulo incluye algoritmos básicos de procesado de imágenes, incluyendo filtrado de imágenes, transformado de imágenesetc.
- Video: Este módulo de análisis de vídeo incluye algoritmos de seguimiento de objetos, entre otros.
- Objdetect: Incluye algoritmos de detección y reconocimiento de objetos para objetos estándar.

6.1.2. Algoritmo propuesto

La idea del algoritmo es hacer que el centro de la imagen coincida con el centro de la región de la cabeza detectada del usuario (ROI o *Region Of Interest*), que se trata de un rectángulo con escala. A continuación se presenta el pseudocódigo implementado:

```
/* Algoritmo de seguimiento de caras */
seguimiento = false
cuentaNoEncontradas = 0
kf = inicializarFiltroDeKalman()
while(1)
  dt = tiempoAhora() - prevt
  img = tomarNuevoFrame()
  if (seguimiento)
    actualizarMatrizDeTransicion()
    estado = kf.predecir()
    cara = estado.rectangulo
    errorX = img.centro.x - (cara.x+cara.width/2)
    errorY = img.centro.y - (cara.y+cara.height/2)
  if(not detectarCaras())
    cuentaNoEncontradas++
    if(cuentaNoEncontradas >50)
      seguimiento = false
      moverCabezaA(0,0)
  else
    cuentaNoEncontradas = 0
    matrizMedidas[] = caraDetectada.rectangulo
    if(not seguimiento) /*primera deteccion*/
      kf = inicializarFiltroDeKalman()
      seguimiento = true
    else
      kf.corregir(matrizMedidas)
    (errorX,errorY) = normalizar(errorX,errorY) /*[-1,1]*/
    posActual = leerControlador()
    refX = (posActualX) + errorX*kP
    refY = (posActualY) + errorY*kP
    (refX,refY) = comprobarLimites(refX,refY)
    publicarAControlador(refX,refY)
```

El algoritmo arranca cuando se detecta al menos una cara (con un umbral de calidad) dentro de la función detectarCaras(), entonces se habilita la variable «seguimiento» y se guarda la detección actual para indicar que en próximas detecciones se intente detectar la misma cara. Esto tiene lugar dentro del método detectarCaras() donde, si «seguimiento» está activa, entocnes se filtran las detecciones para buscar aquella que se asemeje más a la encontrada en la iteración anterior. La razón de esto es que el robot, ante la presencia de varias caras, permanezca siguiendo a la primera con la que se inició.

Un vez se encuentra en seguimiento, las nuevas detecciones se centran en ir actualizando el filtro de Kalman y la predicción de éste es la que se utiliza para computar el error entre la cara detectada y el centro de la imagen.

El hecho de incorporar un filtro de Kalman eleva sustancialmente las prestaciones del seguimiento ya que, aunque no se detecte por algunos instantes ninguna cara, debido a oclusiones, a factores de iluminación, etc, el filtro es capaz de predecir dónde se encontrará ésta debido a que computa el historial de movimientos anteriores. Claro, esto funciona para cortos períodos de tiempo ya que si durante 50 imágenes Clank no ha sido capaz de encontrar ninguna cara entonces se desactiva la variable «seguimiento» y se lleva la cabeza a la posición de reposo (0,0).

En los siguientes apartados se pretende analizar cada etapa del algoritmo más en profundidad.

6.2. Detección

La etapa de detección tiene lugar en la función detectarCaras() y se basa en el uso de la clase «CascadeClassifier» dentro de la librería <objdetect.h>de OpenCv. En la etapa de inicialización se carga un fichero con extensión .xml donde viene reflejado el entrenamiento del clasificador que, en este caso, está enfocado a la detección de caras con distinto escalado.

El método fundamental de la clase es el de «detectMultiScale()» que devuelve un

vector con las caras detectadas y un factor de «calidad» de la detección llamado «levelWeights». Esta etapa de detección se comporta de dos formas distintas según el valor que tome la variable seguimiento:

- Si el seguimiento está activo entonces se busca en el vector que devuelve el método «detectMultiScale()» aquella detección que más cerca se encuentre de la última computada. Se da un umbral de 50 píxeles que conforma un círculo alrededor del vértice superior izquierdo de la detección anterior, y, para cada una de las nuevas, de caer dentro del umbral, se toma como la misma cara y se retorna un valor de verdadero, es decir, que se ha encontrado la cara a la que se está siguiendo.

- Si la variable «seguimiento» toma el valor falso entonces del vector de salida de caras del método «detectMultiScale()» se toma aquella detección que tenga el mayor peso en el vector «levelWeights» y que éste sea mayor a un umbral, que, por defecto, vale 5.0. Esto evita que se detecten caras donde realmente no existen, se minimiza el seguimiento a ruido.

6.3. Filtro de Kalman y Ley de control

Como se adelantaba en la primera sección, en el algoritmo de seguimiento facial se ha integrado un filtro de Kalman con objeto de tolerar oclusiones o pérdidas de detección durante algunos instantes. El vector de estado del filtro se define como 6.1 donde «x» e «y» son las coordenadas del vértice superior izquierdo de la cara detectada en la imagen, v_x e v_y las velocidades de desplazamiento de la detección en píxeles sobre los ejes horizontal/vertical y «w» y «h» que se refieren a la anchura y la altura del rectángulo que envuelve la cara detectada.

$$S = \begin{bmatrix} x & y & v_x & v_y & w & h \end{bmatrix} \tag{6.1}$$

Por otro lado, el vector de nuevas medidas se define como en 6.2, donde «x» e «y» son las coordenadas del vértice superior izquierdo de la cara detectada en la imagen en el instante actual, «w» y «h» son la anchura y la altura del rectángulo que define de la detección actual.

$$Z = \begin{bmatrix} z_x & z_y & z_w & z_h \end{bmatrix} \tag{6.2}$$

Las ecuaciones que expresan la transición entre estados son las que se presentan en 6.3, que, en forma matricial, conforman la matriz mostrada en 6.4. Aquí se asume que el movimiento de las caras detectadas es lineal. Dicha matriz tiene por columnas el vector de estado del instante de tiempo actual y tiene por filas el estado siguiente (transición).

$$\begin{aligned} x_{n+1} &= x_n + vx_n * dt \\ y_{n+1} &= y_n + vy_n * dt \\ vx_{n+1} &= vx_n \\ vy_{n+1} &= vy_n \\ w_{n+1} &= w_n \\ h_{n+1} &= h_n \end{aligned} \tag{6.3}$$

$$A = \begin{bmatrix} 1 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & dt & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

También es necesario definir una matriz que ponga en relación las nuevas medidas con el estado. A esta matriz se la conoce como H 6.5, que tiene tantas columnas como tamaño del vector de estado y tantas filas como tamaño del vector de nuevas medidas, por tanto (4,6).

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

Finalmente, es necesario establecer una matriz de covarianza que quiere expresar el ruido del proceso (reflejado en el estado), se ha optado por los valores mostrados en la matriz 6.6. En el capítulo de resultados tiene lugar la justificación del porqué de esos valores, sobre todo en los de velocidades.

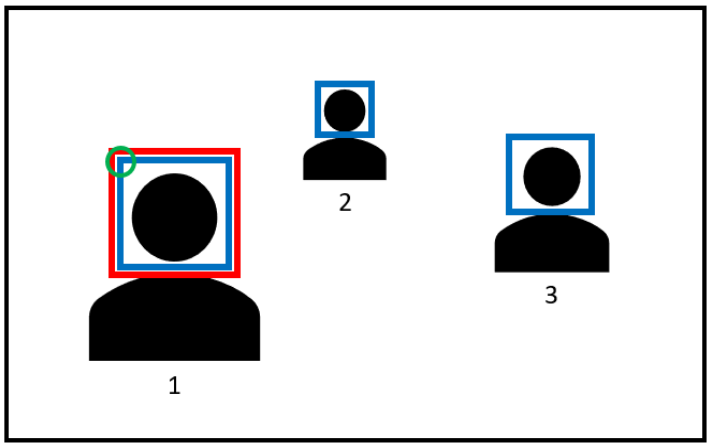
$$Q = \begin{bmatrix} E_x = 1e^{-2} & 0 & 0 & 0 & 0 & 0 \\ 0 & E_y = 1e^{-2} & 0 & 0 & 0 & 0 \\ 0 & 0 & Ev_x = 1e^{-2} & 0 & 0 & 0 \\ 0 & 0 & 0 & Ev_y = 1e^{-2} & 0 & 0 \\ 0 & 0 & 0 & 0 & E_w = 1e^{-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & E_h = 1e^{-2} \end{bmatrix} \tag{6.6}$$

Cuando se está haciendo seguimiento la predicción del filtro se utiliza para obtener el error entre el centro de la cara detectada y el centro de la imagen. Ese error se normaliza entre los valores [-1,1] y se multiplica por una ganancia que actualmente toma el valor de 0.8. Ese incremento se suma o se resta a la posición actual de la cabeza para tender la mirada del robot hacia el usuario.

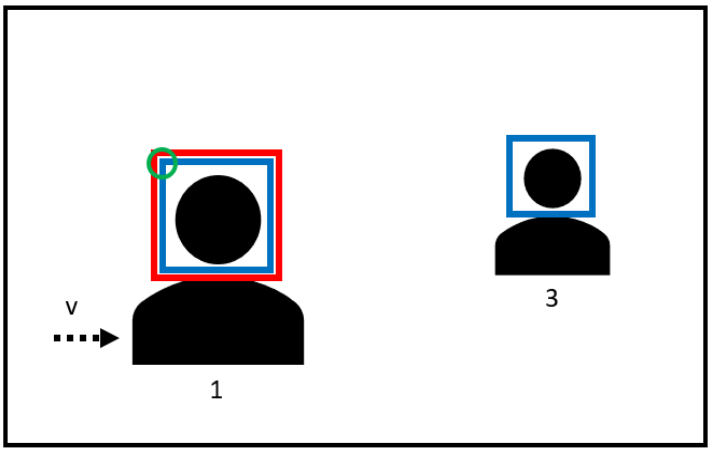
Supongamos que en un instante de tiempo T = n se inicia el seguimiento a un usuario, el denotado por el identificador 1 en la figura 6.2a. El filtro se inicializa, la variable seguimiento se activa y dicha detección se alimenta al filtro.

En otro instante de tiempo posterior, T = n+1, el usuario comienza a desplazarse lateralmente con una velocidad v. Aún en este estado, se detecta la cara del mismo usuario, y, de nuevo, se alimenta la detección al filtro, al que, como en el instante anterior, se le actualiza el incremento de tiempo en la matriz de transición.

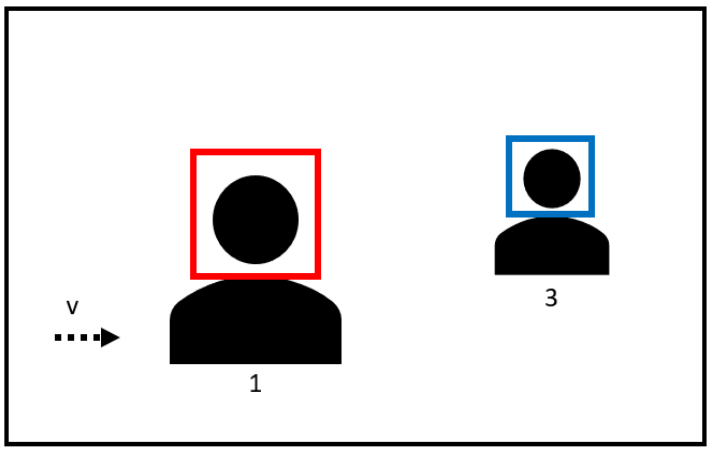
A continuación, se llega a un T = n+2 en el que no se detecta la cara del usuario 1. Sin embargo, el filtro, con la historia del movimiento de éste, es capaz de predecir dónde se ubicará la cara en este último instante. Este proceso es el que se intenta mostrar en las figuras 6.2a, 6.2b, 6.2c.



(a) Instante T=tn



(b) Instante T=tn+1



(c) Instante T=tn+2

Figura 6.2: Ejemplo ilustrativo Filtro de Kalman. En azul las detecciones, en rojo la predición del filtro

6.4. Movimiento de servomotores. Controlador de estado de articulaciones

Cuando se inician la interfaz hardware de Clank y sus controladores, entre ellos se encuentran aquellos específicos de grupos de articulación como piernas, brazos o cabeza. En este caso, se utilizará el controlador de la cabeza, cuyos topics vienen publicados bajo el espacio de nombres «head_controller».

Para enviar comandos hacia los motores se utilizará el topic «/head_controller/command» que lleva asociado un tipo de mensaje «trajectory_msgs::JointTrajectory» con los siguientes campos:

- header: De tipo Header. Aquí se incluye información sobre el sistema de referencia al que va dirijido y el tiempo en el que se genera el mensaje.
- joint_names: Se trata de un vector de cadenas de caracteres. Aquí se especifican los nombres de las articulaciones que se desean accionar. En este caso, las articulaciones que controlan el cabeceo y el alabeo son «head_pan_joint» y «head_tilt_joint».
- points: Se trata de un campo tipo vector de trajectory_msgs/JointTrajectoryPoint. En cada elemento del vector se especifica otro subvector donde se rellenan las posiciones, velocidades o aceleraciones a las que se desea enviar las articulaciones detalladas en el campo anterior. En este caso se utilizarán posiciones y, en cada instante de tiempo serán las obtenidas por el control proporcional.

Las posición máxima y mínima del cada grado de libertad son:

- En alabeo: de -1.3 radianes a 1.3 radianes
- En cabeceo: de -0.3 radianes a 0.3 radianes

Esta saturación es la que se comprueba en el método comprobarLimites() previamente mencionado en la sección del algoritmo propuesto.

Por otro lado, para realimentar la posición actual de los servomotores se hace uso del topic «/head_controller/state», de tipo «control_msgs::JointTrajectoryControllerState» que contiene los siguientes campos:

- header: De tipo Header. Aquí se incluye información sobre el sistema de referencia al que va dirijido y el tiempo en el que se genera el mensaje.
- joint_names: De nuevo, las articulaciones que controlan el cabeceo y el alabeo son «head_pan_joint» y «head_tilt_joint».
- desired: con el tipo trajectory_msgs/JointTrajectoryPoint refleja las posiciones/velocidades y aceleraciones deseadas.
- actual: de tipo Con el tipo trajectory_msgs/JointTrajectoryPoint que devuelve la posición actual en la que se encuentran las articulaciones. Es precisamente este campo el que se tiene en cuenta a la hora de realimentar la posición de los servomotores en el sistema de seguimiento facial.
- error: con el tipo trajectory_msgs/JointTrajectoryPoint expresa el error entre la posición deseadaa y la actual para cada articulación.

Capítulo 7

ANÁLISIS ECONÓMICO DEL PROYECTO

El proyecto abarca la creación de un robot de bajo coste, tanto a nivel mecánico, electrónico y de software. En el presente capítulo se pretenden computar los costes asociados a estos aspectos así como al personal que le ha llevado a cabo y así ofrecer una estimación del coste asociado al proyecto.

En este apartado han sido tenidos en cuenta otros trabajos como referencia [5].

7.1. Recursos empleados

Como en cualquier proyecto de ingeniería existen una serie de recursos tanto físicos (hardware, mecánica) como programas informáticos (software). Este apartado incluye ordenadores, sensores, actuadores, electrónica y programas utilizados. 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:

- Mecánica: Impresora 3D AnyCubic prusa i3 y Rollo de material de impresión en

plástico PLA.

- Electrónica-sensórica: Cámara Raspberry Pi 3, minimu Pololu v5.
- Electrónica-microcontroladores: Raspberry Pi modelo 3b+ y placa OpenCM9.04 modelo B.
- Electrónica-actuadores: 18 servomotores Dynamixel AX-12 y 2 servomotores tipo XL-320.
- Sistemas Operativos empleados: Ubuntu 16.04 y ROS(Robot Operating System) Kinetic y Raspbian.
- Software empleado: FreeCad para el diseño 3D, Editor de texto Office Writer.

7.2. 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.

7.2.1. Costes de personal

La realización del proyecto ha sido llevada a cabo por un ingeniero al que ha llevado las tareas de búsqueda de información previa, diseño e implementación del robot.

COSTE ANUAL

Sueldo Bruto	28.885,92	€
Seguridad social (35 % sueldo)	10.110,07	€
Total	38.995,99	€

Tabla 7.1: *Salario anual*

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[14] (tablas 7.1 y 7.2).

[1]Salario convenio Euros/mes por 14 pagas ingeniero es 2.063,28 €[13]

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 7.2: *Días Efectivos por año*

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

$$220,89\frac{\text{días}}{\text{año}} \cdot 8\frac{\text{horas}}{\text{día}} = 1{,}767,12\frac{\text{horas}}{\text{año}} \tag{7.1}$$

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

$$\frac{38{,}995,99\frac{€}{\text{año}}}{2{,}063,28\frac{\text{horas}}{\text{año}}} = 18{,}90\frac{€}{\text{hora}} \tag{7.2}$$

En la siguiente tabla (tabla 7.3)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 7.3: *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 del ingeniero por el coste por hora calculado anteriormente:

$$355 \quad \text{horas} \cdot 18{,}90\frac{€}{\text{hora}} = 6709,5€ \tag{7.3}$$

COSTE PERSONAL DIRECTO	6709,5 €

7.2.2. 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 16.04, 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 Asus ROG Scar Strix: vida útil: 6 años. Factor de corrección por amortización 0.23.
- Microsoft Windows 10 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 7.4:

MATERIAL	IMPORTE	FACTOR	AMORTIZACIÓN ANUAL
Microsoft Windows Vista Profess.	139,0 €	0,3	41,7 €
Ordenador sobremesa HP Compaq	900,0 €	0.23	207,0 €
Total material	**1039,0** €		**248,7** €

Tabla 7.4: *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:

$$\frac{248,7\frac{€}{\text{año}}}{1{,}767,12\frac{\text{horas}}{\text{año}}} = 0,14\frac{€}{\text{hora}} \tag{7.4}$$

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:

$$355 \quad \text{horas} \cdot 0,14 \frac{€}{\text{hora}} = 49,7€ \tag{7.5}$$

COSTES DE AMORTIZACIÓN DE RECURSOS OFICINA	49,7 €

7.2.3. Costes de equipamiento físico

En estos costes se incluyen tanto el coste de los elementos sensores, electrónica y mecánicos utilizados en Clank para este proyecto:

MATERIAL	IMPORTE
Servomotores (x20)	875,65 €
Impresora AnyCubic Prusa i3	349,98 €
Rollo impresión PLA	20,00 €
Raspberry Pi 3b+	36,40 €
OpenCM 9.04	19,98 €
MiniImu v5	20,45 €
Cámara Raspberry Pi 3b+	26,90 €
Kits de tornillería	15,75 €
Total equipamiento físico	**1.365,11** €

Tabla 7.5: *Costes del equipo físico*

COSTES DE EQUIPAMIENTO FÍSICO	1.365,1 €

7.2.4. Costes directos totales

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

$$6709,5€ + 49,7€ + 1{,}365,11€ = 8{,}124,31€ \tag{7.6}$$

COSTES DIRECTOS	8.124,3 €

7.3. 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 7.6: *Costes Indirectos*

COSTES INDIRECTOS	**320,0 €**

7.4. 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	8.124,31 €
Costes indirectos	320,00 €
Coste total del proyecto	**8.444,31** €

Tabla 7.7: *Costes Totales*

COSTE TOTAL DEL PROYECTO	**8.444,3 €**

Capítulo 8

RESULTADOS OBTENIDOS

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

8.1. Hardware y Mecánica

Enmarcada entre los hitos principales del proyecto se encontraba la creación de una plataforma robótica humanoide de bajo coste, con, al menos, 18 grados de libertad. La impresión aditiva se ha mostrado como una tecnología capaz de afrontar con solvencia el diseño de prototipos. En este sentido, el material PLA utilizado en el robot Clank cumple con los requerimiento de consistencia, siendo difícil que con el torque que presentan los servomotores o por golpes fortuitos el robot parta por alguna de las articulaciones.

Pese a esto, como en cualquier proyecto de ingeniería, durante la primera fase de pruebas con el robot, un par de piezas (correspondietes a la cadera y la rodila) han tenido que ser reimprimidas debido a que se han roto fruto del incumplimiento de los límites mecánicos del robot. Sin embargo, aquí emerge otra gran ventaja del diseño del robot y la impresión 3D, que es la facilidad para sustituir piezas, debido a su modularidad. Una lección aquí aprendida es que la orientación con la que se imprime la pieza debe ser tal que la dirección del esfuerzo que vaya a soportar la pieza sea lo más perpendicular posible a la dirección de crecimiento de las capas, para garantizar

una mayor durabilidad.

En cuanto a los sensores utilizados, tanto el sensor inercial IMU como la cámara han mostrado ser de gran utilidad para el desarrollo de la percepción del robot. El ordenador de a bordo, la Raspberry Pi 3, ha desmostrado ser un microcontrolador capaz de soportar las tareas asignadas, basicamente tareas de bajo y medio nivel donde entran recolección de datos y gestión de órdenes a actuadores, es decir, la interacción con el hardware y comunicaciones con el ordenador ROS externo. Todas estas tareas en la Raspberry daban como resultado un por centaje de uso de la CPU de, aproximadamente, el 55 %.

En cuanto a los actuadores, se analizan los 20 dynamixel con los que cuenta el robot. En este punto cabe decir que aquellos 18 correspondientes al modelo AX-12 cumplen con una buena relación calidad precio, aunque no es todas las partes del cuerpo trabajn con la misma solvencia a la hora de cumplir su función. Se ha observado que para los brazos, estos servomotores son suficientes para desempeñar una tarea de control. Sin embargo, los que están ubicados en las piernas, si bien han sido suficientes para cumplir con las funciones de control del presente proyecto, la respuesta del control que se requiere en comportamientos más avanzados como por ejemplo, patrones de caminar, no es suficiente, por tanto se recomendarían los correspondientes al siguiente modelo, los Dynamixel RX-28. Por su parte, los servomotores de menor potencia XL-320 ubicados en la cabeza, también cumplen con creces con su tarea asignada.

Finalmente, respecto a la estética y diseño de Clank se tiene que destacar que el robot, en su diseño final, presenta un aspecto amigable que tiende a hacer que el usuario muestre interés y se acerque al robot, lo que es un aspecto positivo en la interacción humano-robot.

8.2. Respecto del sistema postural

El sistema postural se presenta como uno de los aspectos de mayor importancia a la hora de incrementar la interacción humano-robot, a la vez que, con él, se aprovechaban las capacidades en cuanto a grados de libertad que posee el robot.

Para probar el sistema, se ha diseñado el experimento de llevar al robot a diferentes

posturas, tanto predefinidas en el asistente Moveit, el Moveit Setup Asistant, como de planificación libre desde los «markers» de Rviz.

Cuando se trata de alcanzar posturas con los brazos o cabeza, se llega la posición objetivo con buena precisión (+-0.01 radianes), de manera repetible. Por otro lado, cuando el movimiento implica posturas más forzadas para el robot como es el gesto de levantarse y sentarse, en determinadas ocasiones el robot no es capaz de alcanzar las mismas posiciones finales con una alta repetibilidad, auqnue se considera normal debido al elevado peso del robot con respecto al torque máximo de los servomotores.

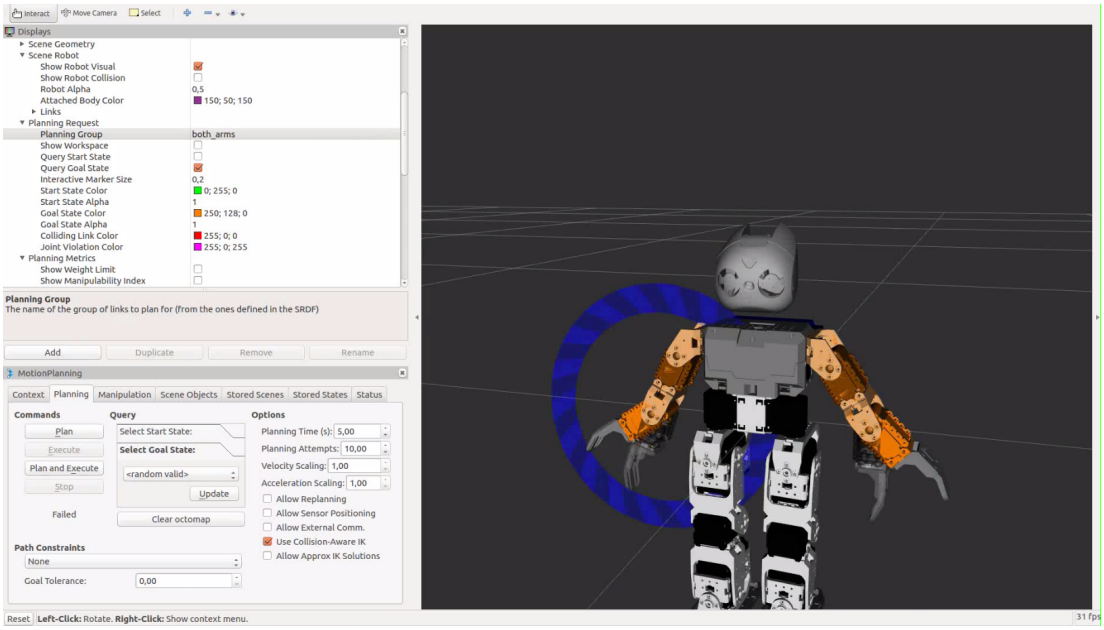


Figura 8.1: Planificador postural con «markers» en Rviz

En la figura 8.1 se muestra el planificador Moveit a través de la interfaz gráfica Rviz. En este caso, el grupo de planificación son los dos brazos y en naranja se dibujan las posiciones objetivo a las que se desean mandar los brazos y en blanco, negro y gris se indica la posición actual del robot.

8.2.1. Validación

La nueva línea de actualización y desarrollo de paquetes de ROS_industrial han sido claves para lograr otro de los objetivos iniciales, esto es, la planificación y control de todos los grupos de articuaciones. Con la descripción del robot Clank en URDF y

estas herramientas se ha llegado a cumplir con creces el objetivo, teniendo una opción gráfica para planificar o para hacerlo mediante la API de MoveGroup (en Python MoveitCommander) a través de un script de Python. Se da por válida la consecución del objetivo principal y del objetivo secundario de adoptar el uso del estándar ROS Industrial.

8.3. Respecto del sistema de estabilidad de balanceo

Para probar el trabajo conjunto del sensor de inercia y los servomotores se ha diseñado el experimento de someter a Clank a un plano inclinado en ambos sentidos y observar la respuesta del control.

Primeramente, cuando el robot se encuentra estático, en la figura 8.2 se analiza el comportamiento del giróscopo. Como se puede observar, existe un rizado normal en la señal de los ejes de giro. Sin embargo, el módulo máximo de éste no supera los 0.1 rad/s y su media es prácticamente cero, por lo que este proceso de calibrado que se lleva en la placa OpenCM y que se alimenta a la Raspberry, es correcto. Cabe decir que esta señal será posteriormente introducida en un vector cuya media de todos los valores será la utilizada para actualizar la transformada entre el sistema odometría y el sistema de referencia imu del robot.

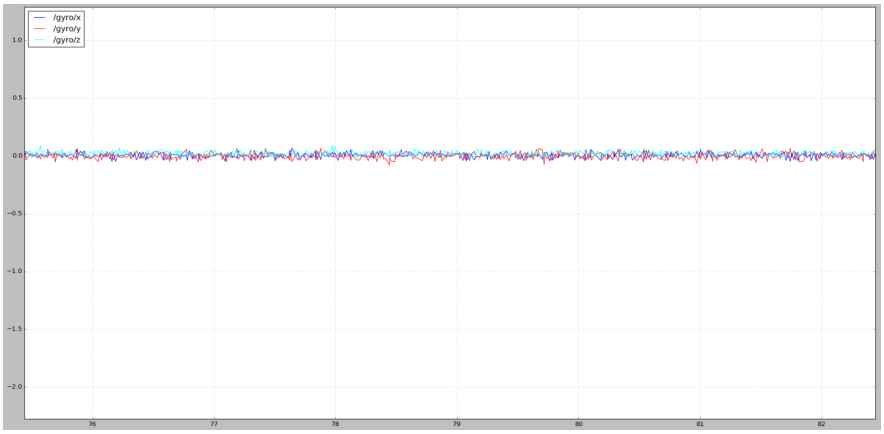
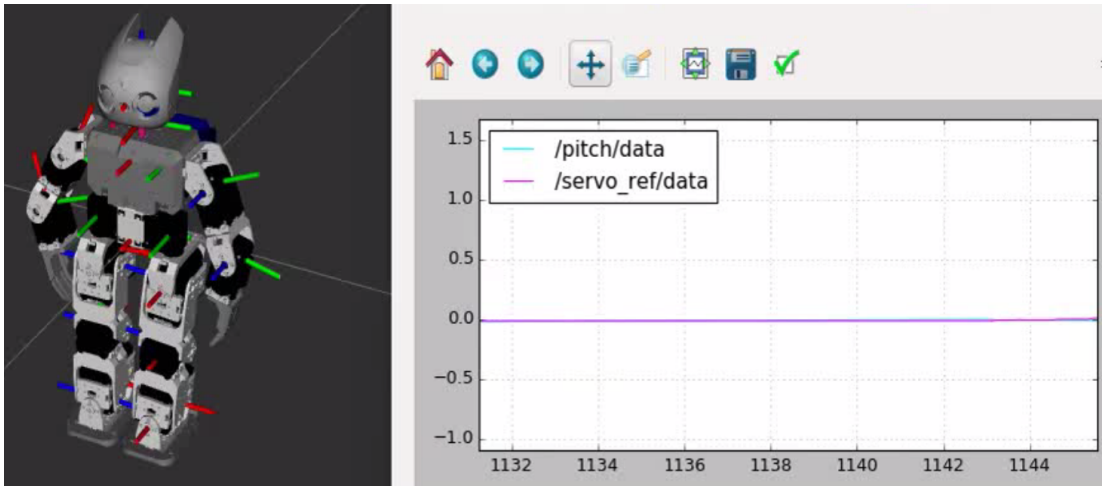


Figura 8.2: Datos del giróscopo después del calibrado en la placa OpenCM

A continuación se ha llevado el plano inclinado hacia el frente. Esta transición se muestra en las sigueintes figuras. En la gráfica de la figura se indican, en azul, el valor

del ángulo devuelto por el IMU en cada instante de tiempo y, en rosa, la referencia de posición que se envía al servomotor para hacer que, precisamente, dicho ángulo tienda a cero ante las perturbaciones en el plano de cabeceo.

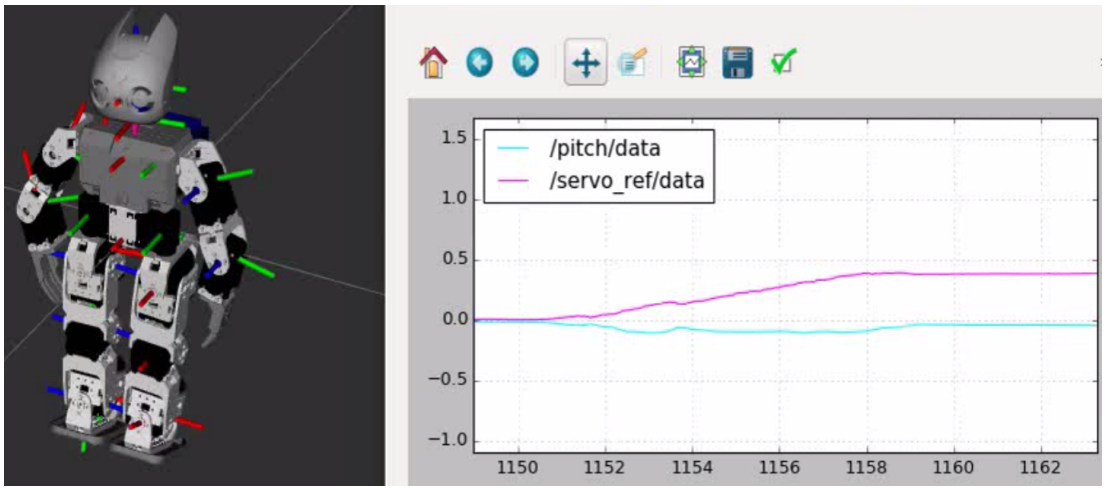
(a) Clank en posición de inclinación cero



(b) Clank en Rviz y métricas cero inclinación

Figura 8.3: Inclinación cero en el plano de cabeceo

(a) Clank en posición de máxima inclinación



(b) Clank en Rviz y métricas de máxima inclinación

Figura 8.4: Máxima inclinación en el plano de cabeceo en un sentido

8.3.1. Validación

El sistema de balanceo se muestra con un resultado funcional. El robot corrige adecuadamente cuando no se perturba el sistema a una frecuencia elevada. De lo contrario,

el control se hace inestable. Esto es debido a tres factores:

- El primero, el tiempo en la respuesta de los servomotores. Como se ha comentado en la primera sección del capítulo, los servomotores AX-12 tienen una capacidad de torque justa para soportar el torque de mantener el robot erguido, con el peso que tiene.

- El segundo, la deriva existente en el IMU el cual, cuanto más tiempo se deja en funcionamiento, el error respecto de la vertical puede aumentar.

- En tercer lugar, para el control de estabilidad se ha asumido que el lazo interno del PID de los servomotores tiene una respuesta ideal, sin tener en cuenta que el factor dinámico comienza a ser importante cuando el servo intenta llevar la articulación a la referencia deseada. Sobre todo en este caso de los servomotores de los tobillos del robot.

Por tanto, se debería concluir que el objetivo de lograr un sistema de equilibrio en plano de cabeceo ha sido cumplido bajo las condiciones anteriormente expuestas.

8.4. Respecto del sistema de seguimiento facial

La librería OpenCv junto con ROS ha probado ser una buena combinación sobre el que construir el sistema de seguimiento. Para demostrarlo se ha llevado a cabo e experimento de hacer que Clank siga con la mirada a un usuario, inclusive con más de uno en la imagen (hasta tres personas) y provocando oclusiones en el seguimiento.

La inclusión del filtro de Kalman ha sido fundamental a la hora de hacer el sistema más tolerante/robusto y generar un comportamiento más natural en el movimiento de la cabeza.

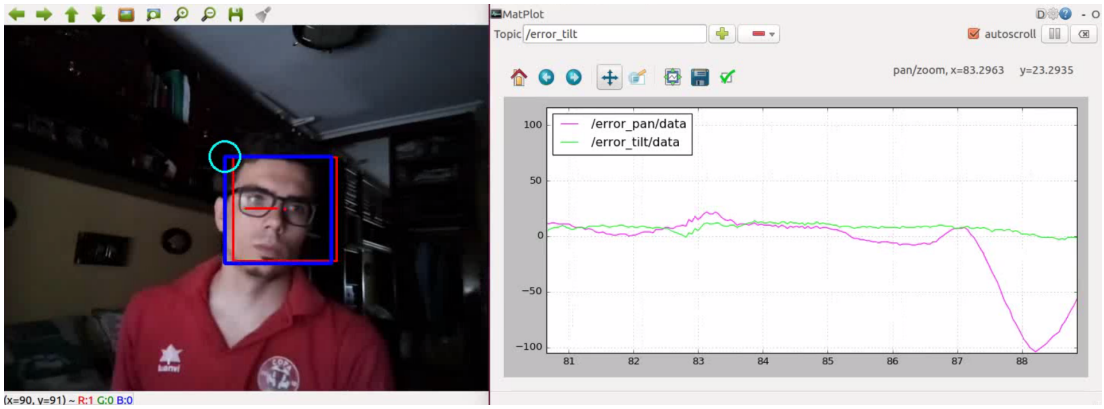


Figura 8.5: Seguimiento del usuario

En la figura 8.5 se muestran las variables fundamentales para comprender este sistema. A la izquierda se indica, en un recuadro azul, la detección del usuario, en un recuadro rojo la predicción del filtro, y en líneas roja y verde los errores en los ejes x e y, que son la diferencia entre la predicción del filtro y el centro de la imagen. En la gráfica de la derecha se muestran dichos errores, en píxeles, a lo largo del tiempo. Estos ultimos se vuelven distintos de cero (aunque en realidad nunca son cero totalmente) con el movimiento del usuario.

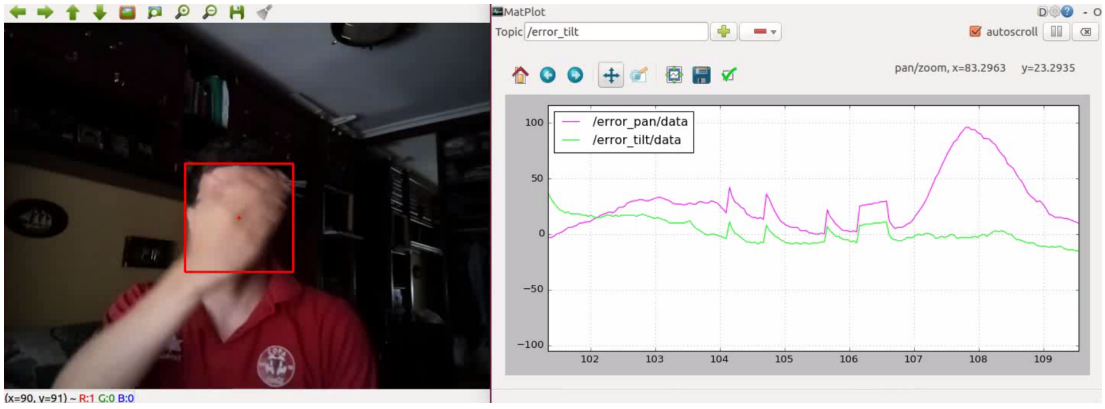


Figura 8.6: Seguimiento ante oclusión

Por otro lado, en la figura 8.6 se muestra cómo ante la oclusión temporal del usuario, el sistema sigue adelante, con la predicción del filtro. En este último caso, si la oclusión es mantenida (más de 50 frames) estonces se considera usuario perdido, se vuelve la cabeza a la posición de reposo y se emprende una nueva identificación de usuarios.

8.4.1. Validación

Con el objetivo de mejorar la interacción humano-robot en el primer capítulo se marcaba el hito de lograr un seguimiento de caras lo más natural posible. Los resultados obtenidos prueban que en efecto, el impacto es positivo y la refuerza, siendo la trasición de movimiento de la cabeza suave sin oscilaciones. Por tanto se da el objetivo como validado.

Capítulo 9

CONCLUSIONES Y FUTURAS LÍNEAS DE DESARROLLO

Este último capítulo se dedica a exponer las conclusiones extraídas del desarrollo del proyecto, así como nuevas mejoras y líneas futuras de desarrollo.

9.1. Conclusiones

La robótica humanoide bípeda sigue siendo un reto en la actualidad. Muchos son los esfuerzos que se están poniendo en investigar este tipo de plataformas que, debido a su complejidad, requieren de integrar muy bien distintas disciplinas dentro de la ingeniería como son mecánica, electrónica y control.

El principal objetivo del proyecto ha sido el de crear una plataforma robótica en la que integrar distintas tecnologías, todo ello usando ROS como entorno de trabajo. Dicha descripción se enmarca perfectamente en la robótica humanoide y es cómo se decide apostar por una plataforma robótica y modular de este estilo.

El presupuesto con el que se partía era limitado, el coste de materiales no debía ser muy elevado por lo que la elección para la fabricación del robot ha sido mediante impresión 3D. Por otro lado, la electrónica utilizada también es de bajo coste, aquí se incluyen Raspberry Pi, IMU y OpencM, siendo los 20 servomotores lo más caro del

proyecto.

Para abordar la interacción humano-robot se han desarrollado distintos módulos software que operan en distintas áreas del robot, como son la visión artificial, el control y la planificación. ROS ha sido un Middleware acertado sobre el que basar el desarrollo de estos sistemas así como la adopción del estándar ROS Industrial.

En lo que refiere a la visión artificial se ha propuesto un sistema de seguimiento facial por imagen, las librerías utilizadas han sido OpenCv. Los resultados finales de este módulo de seguimiento han sido muy satisfactorios, apoyando perfectamente la tarea de mantener la mirada en el usuario, haciendo que éste se muestre más receptivo y abierto a acercarse al robot, creando un buen clima para interactuar.

Respecto del área de control, se ha desarrollado un sistema de equilibrio en el plano de cabeceo. Para abordar el problema se ha uilizado un sensor de inercia o IMU, con lo que se obtiene la orientación del robot y, para actuar, se utilizan los servomotores ubicados en los tobillos. Como resultado se ha obtenido una buena respuesta del robot ante perturbaciones de baja frecuencia. Aunque aquí también se concluye que, para desarrollar sistemas de equilibrio más complejos se requieren servomotores que desarrollen más par, sobre todo, los ubicados en las piernas que soportan el peso y par resistente de todo el robot.

Finalmente, para aprovechar la movilidad y grados de libertad del robot, un planificador para cada grupo de articulaciones es la solución adoptada. En este punto se han intergrado las librerías y herramientas de las que dispone el nuevo ROS Industrial, como son Moveit o ROS control. El resultado obtenido ha sido excepcional pudiéndose utilizar Rviz para comandar posturas destino de una o varias articulaciones a la vez o mediante las interfaz de programación de Python (MoveitCommander) y C++ (MoveGroup).

9.2. Futuras líneas de desarrollo

A partir del desarrollo propuesto en este proyecto y como líneas futuras de investigación se proponen varias alternativas por las que seguir mejorando y completando a Clank, el robot desarrollado en este proyecto.

Con el uso de unos servomotores más potentes en las piernas se puede plantear un control de equilibrio bípedo más avanzado que incluya patrones de caminar o de salto. Para coordinar a alto nivel todos los subsitemas y estados del robot se propone el uso de árboles de comportamiento (máquinas de estado avanzadas), de esta forma se pueden gestionar, por ejemplo, que cuando el robot se haya caído o haya perdido el equilirio, entonces éste desarrolle, comportamientos de recuperación. Otro aspecto que refuerza la interacción del robot con el usuario puede ser la incorporación de un sistema de chatbot y reconocimiento de voz.

El presente proyecto se ha desarrollado, mayormente, en clave de reforzar la interacción humano-robot. Sin embargo, el aspecto de la interacción robot-entorno admite muchas otras mejoras. Mediante el estudio avanzado de la odometría en este tipo de robots unido a la incorporación de sensores que proporcionen posición del robot respecto de un entorno se puede desarrollar un sistema de localización, y, por ende, de navegación.

Referencias

[1] Masahiro Mori. The uncanny valley. In *Energy, vol. 7, no. 4*, 1970.

[2] J Edward, Witaya Wannasuphoprasit, and Michael Peshkin. Cobots: Robots for collaboration with human operators. 03 1999.

[3] Md Akhtaruzzaman, A.A. Shafie, and Mahbub Rashid. Designing an algorithm for bioloid humanoid navigating in its indoor environment. *Journal of Mechanical Engineering and Automation*, 2:36–44, 05 2012.

[4] Roberto Pinillos Herrero. Diseño de un robot social y validación de servicios en entornos hoteleros. 2014, Trabajo Fin de Master. Universidad de Valladolid.

[5] Mario Domínguez López. Sistema de interacción visual para un robot social. 2017, Trabajo Fin de Grado. Universidad de Valladolid.

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

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

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

[9] 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.

[10] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[11] ROS Actionlib, Online, Visitado el 15 de Julio de 2019 `http://wiki.ros.org/actionlib`.

[12] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Fernández Perdomo. ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software, 2 (20), pp.456 - 456*, 2017.

[13] Convenio del Metal Valladolid, Online, Visitado el 15 de Agosto de 2019 `https://www.cntvalladolid.es`.

[14] Alexis Juárez Sánchez. Desarrollo de un sistema de visón-3d para su integración en un robot móvil social. 2014, Trabajo Fin de Grado. Universidad de Valladolid.

[15] José Mayoral Baños, Jesús Aarón Galicia Valdovinos, Arturo Parrales Salinas, and Luis Martín Contreras Tapia. Análisis cinemático del robot humanoide: Darwin op. *Primer Concuso de Investigación, Desarrollo e Innovación Tecnológica (ULSA 2012)*, 05 2012.

[16] Jonathan Alexander Soto-Montoya, Julio César Gómez-Naranjo, and Jaime Alberto Guzmán-Luna. Análisis del comportamiento del robot humanoide bioloid en situación específica. *Lámpsakos*, page 66, 01 2014.