VERSION 2.10.1

Robot Operating System (ROS)

R. Suárez[a], J. Rosell[a], M. Vinagre[b], F. Cortes[b], A. Ansuategui[c], I. Maurtua[c],
D. Martin[d], A. Guash[d], J. Azpiazu[f], D. Serrano[g], N. García[g].

[a] Intitut d'Organització i Control de Sistemes Indusrials (IOC), Universitat Politècnica de Catalunya (UPC)
[b] Leitat Technological Center
[c] Fundación Tekniker
[d] Pilz Industrieelektronik S.L.
[f] Fundación Tecnalia Research & Innovation
[g] Fundació Eurecat

Este trabajo se ha llevado a cabo por iniciativa y en el seno del Grupo de Trabajo de Innovación de la *Asociación Española de Robótica y Automatización* (AER).

ÍNDICE

Robot Operating System (ROS)

1. Introducción

El sector comercial de la robótica está dominado por sistemas cerrados donde se prioriza la dependencia del proveedor sobre la innovación. Cada robot dispone de su propio sistema operativo y lenguaje de programación. Esto significa que para programar un robot se debe escribir código que no es reutilizable en otros tipos de robots, hay que aprender APIs nuevas, y enfrentarse a cuestiones complejas relacionadas con la escritura de código de bajo nivel. Mantener la tecnología basada en sistemas cerrados obstaculiza los avances en el campo académico, obligando a los investigadores a dedicar mucho esfuerzo en la puesta en marcha del hardware que compone su banco de ensayos, y, por tanto, dejando poco tiempo para la innovación. Ha habido varios esfuerzos para producir herramientas de código abierto para la investigación y el desarrollo en el campo de la robótica. Entre ellas cabe destacar los proyectos Carmen[1], Urbi[2] o Player/Stage[3], pero ninguna consiguió una adopción generalizada por parte de investigadores y desarrolladores de robótica. Hasta que llegó ROS, acrónimo de *Robot Operating System* (Sistema Operativo para Robots).

Dos estudiantes de doctorado de la Universidad de Standford, Keenan Wyrobek y Eric Berger estaban empezando sus doctorados y se enfrentaron a esa situación de reinvención de la rueda: quienes se proponían innovar en el software de robótica dedicaban el 90% de su tiempo a reescribir el código que otros habían escrito antes para construir un prototipo de banco de pruebas, y solo el 10% de sus esfuerzos se dedicaba a la innovación. ROS fue su propuesta para hacer frente a esa enorme cantidad de tiempo perdido. En 2006 plantearon un plan para encontrar donantes que financiaran la construcción de 10 robots idénticos, llevarlos a 10 universidades y junto a un equipo de ingenieros de software construir el software base común y las herramientas para desarrolladores que permitieran compartir y reutilizar los avances de los demás. Utilizando aportaciones de la propia universidad y aportaciones individuales (como por ejemplo la recibida de Joanna Hoffman y Alain Rossmann, miembros del primer equipo de Apple Macintosh) construyeron el robot PR1, que sirvió para conseguir el apoyo al proyecto de los principales equipos de I+D de software de robótica de todo el mundo.

Mientras Keenan y Eric luchaban por hacer realidad su ambiciosa visión de hacer de ROS el "Linux de la robótica" conocieron a Scott Hassan, inversor y fundador de Willow Garage, un centro de investigación centrado en productos de robótica. Scott había creado empresas revolucionarias de Internet (Google y eGroups) utilizando software de código de abierto y quería que los futuros empresarios de la industria de la robótica tuvieran una base similar de código abierto. A Scott le pareció tan interesante la idea que en 2008 decidió financiarla y empezar con ellos un Programa de Robótica Personal dentro de Willow Garage. Así, en 2009 nació el robot PR2 y un año más tarde, en 2010, la primera versión de ROS. El proyecto ROS llegó a ser tan importante que todos los demás proyectos de Willow Garage fueron descartados y Willow Garage se concentró únicamente en el desarrollo y la difusión de ROS.

Los promotores de aquella iniciativa eran conscientes de que el PR2 no sería el único robot del mundo en usar ROS, ni siquiera el más importante, y, en el deseo de que ROS fuera útil para otros robots, se esforzaron en definir niveles de abstracción que permitieran reutilizar gran parte del software. Esto condujo a la adopción de ROS en una variedad sorprendentemente amplia de robots, así como en dominios que van más allá de la comunidad de investigación académica en la que se centraron inicialmente. Los siguientes años superaron todas las expectativas: los logros en el campo de la robótica se compartían de forma reproducible en ROS, la Fundación de Robótica de Código Abierto (OSRF, de su nombre en inglés *Open Source Robotics Foundation*) se convirtió en el administrador de ROS tras el cierre de Willow Garage en 2014, la comunidad de ROS creció exponencialmente e incluso la industria comenzó a participar en la iniciativa. De cara a satisfacer mejor las necesidades de una comunidad ROS

[1] http://carmen.sourceforge.net/
[2] https://web.archive.org/web/20100512123920/http://www.urbiforge.org/
[3] http://playerstage.sourceforge.net/

más amplia, abordando sus nuevos casos de uso, la OSRF se esforzó en crear ROS2, como un conjunto paralelo de paquetes que pudieran instalarse junto a ROS1 (ROS original) e interactuar con él.

Tras estos primeros años de andadura, ROS puede considerarse el *framework* de facto para el desarrollo e investigación de tecnologías robóticas. Además, la popularidad de ROS ha seguido creciendo en la industria con el apoyo de proyectos como ROS-Industrial (ROS-I), una iniciativa de código abierto que extiende las capacidades avanzadas de ROS a hardware y aplicaciones industriales relevantes. ROS es también una comunidad global de código abierto formada por ingenieros, desarrolladores y aficionados que contribuyen a que los robots sean mejores y más accesibles. Cada año se celebran eventos ROSCon en los que desarrolladores y entusiastas de la robótica se reúnen para aprender cosas nuevas, presentar sus proyectos y relacionarse entre sí. Incluso hay varios eventos nacionales oficiales como la ROSCon Japan o la ROSCon France.

ROS está aquí para quedarse. ROS y la comunidad que lo rodea siguen creciendo, se han consolidado en la comunidad científica y la aún débil presencia en el ámbito industrial puede cambiar en breve con la adopción de ROS-I por los principales fabricantes en el campo de la robótica.

2. Características y prestaciones

2.1. Infraestructura de comunicaciones y herramientas de creación de software

Un *middleware* es un software que provee la infraestructura de comunicaciones necesaria para conectar componentes o aplicaciones de software. El núcleo de ROS es un *middleware* orientado a sistemas robóticos que permite el intercambio de datos de manera consistente entre diferentes aplicaciones a través de un canal común, facilitando el desarrollo de sistemas de computación distribuidos. De esta manera, ROS cubre la necesidad de comunicación entre los múltiples procesos involucrados en un sistema robótico, que pueden estar ejecutándose en el mismo computador o no. ROS se ha convertido en un estándar *de facto* para la interoperatividad de software en robótica.

Las aplicaciones de ROS están compuestas de varios procesos independientes, débilmente acoplados mediante su interconexión como nodos de un grafo (*ROS runtime graph*) y que comparten información a través de comunicaciones fuertemente "tipadas" (es decir, que no permiten violaciones de los tipos de datos). Por un lado, esta estructura facilita el desarrollo de las aplicaciones ya que cada nodo expone una API (*Application Programming Interface)* mínima al resto del grafo y, por el otro, da robustez al sistema ya que las posibles caídas informáticas quedan aisladas. La gestión del grafo ROS (la negociación de las conexiones y el registro y búsqueda de los recursos del grafo) la lleva a cabo el programa ROS *Master*. Existe además un *Parameter Server* que gestiona parámetros persistentes que pueden ser consultados por todos los nodos.

Los mecanismos de comunicación ofrecidos por ROS son: a) Paso de mensajes mediante *Publicador/Subscriptor* anónimos, a través de buses nominales denominados ROS *topics*, que permite una comunicación asíncrona entre el nodo publicador que emite un mensaje y el conjunto de nodos subscriptores que lo reciben (conjunto que puede ser vacío); b) Llamada remota de procedimientos mediante *Cliente/Servidor*, que permite una comunicación síncrona entre el nodo cliente que llama y el nodo servidor que responde; c) Llamada remota de procedimientos (con posibilidad de interrupción) mediante *Acciones*, que permite una comunicación tipo *Cliente/Servidor,* pero con una realimentación que facilita la monitorización de su progreso, y que incluye la posibilidad de interrupción de la llamada.

Todo el software de ROS está implementado en C++, aunque hay también interfaces en Python, y se organiza en paquetes, que son colecciones coherentes de ficheros, tanto ficheros fuente como de soporte, y que tienen un propósito específico. ROS ofrece, por un lado, *catkin*[4] la herramienta para la creación de software desarrollado en ROS (especificación de rutas, dependencias, compilación, enlace e instalación), que permite la generación de los

[4] http://wiki.ros.org/catkin

nodos definidos en los paquetes. Por otro lado, ofrece un mecanismo, *roslaunch*[5], para ejecutar todos los nodos involucrados en el sistema y que configuran el grafo ROS.

2.2. Herramientas básicas para el desarrollo de aplicaciones robóticas

Para facilitar el desarrollo de aplicaciones robóticas, ROS ofrece:

- Definiciones estándar de mensajes útiles para sistemas robóticos: mensajes de conceptos geométricos (e.g. transformaciones y vectores), de sensores (e.g. cámaras y láseres), o de datos de planificación y navegación (e.g. trayectorias y mapas), entre otros. Existen, por ejemplo, los mensajes tipo *Image* o *PointCloud* para las imágenes 2D y 3D, los de tipo *JointState* para los valores de las articulaciones del robot, o los de tipo *Transform* para los sistemas de referencia de los eslabones. Estos dos últimos son usados, respectivamente, por los nodos *joint_state_publisher* y *robot_state_publisher*, disponibles en los correspondientes paquetes, para publicar el estado del robot.
- Biblioteca Geométrica (*tf*): biblioteca con las herramientas necesarias para gestionar (definir y relacionar) las transformaciones involucradas en un sistema robótico, tanto las estáticas (por ejemplo, las de una cámara fija a una base), como las dinámicas (por ejemplo, las de las articulaciones de un brazo robot).
- Lenguaje para modelar robots (*Unified Robot Description Format*, URDF): formato de fichero XML con la información de los eslabones y articulaciones de la cadena cinemática que forma el robot, más flexible que los parámetros DH (Denavit-Hartenberg) usualmente utilizados, y que puede incluir información adicional como la localización de sensores, la apariencia visual de cada parte del robot, o las características dinámicas de las articulaciones y de los eslabones. El modelo en URDF del robot se almacena como parámetro (*robot_description*) en el *Parameter Server*. También se dispone de un modelo semántico del robot (*Semantic Robot Description Format*, SRDF) que describe información semántica del robot que no está incluida en el fichero URDF.
- Visor *Rviz*[6]: Herramienta de visualización que permite mostrar el robot e información sensorial como imágenes 2D y 3D. Es un nodo del grafo ROS que lee el modelo del robot en el *Parameter Server* y que se subscribe a los mensajes publicados con la información del estado del robot y a los mensajes publicados por los nodos que controlan las cámaras.
- Simulador *Gazebo*[7]: Herramienta de simulación que tiene un motor de simulación dinámica potente, gráficos de alta calidad, y buenas interfaces gráficas y de programación. Existe como aplicación aislada y ha sido integrada como paquete de ROS (*gazebo_ros_pkgs*[8]). Mediante el uso de *plugins, Gazebo* permite simular sensores[9] y el comportamiento del robot actuado a través de la librería *ros_control*[10].

2.3. Herramientas de alto nivel para el desarrollo de aplicaciones robóticas

2.3.1. *Percepción*

Los sistemas robóticos "inteligentes" deben ser capaces de ejecutar las acciones más adecuadas en situaciones no previstas. Para ello, el robot debe ser capaz de percibir, comprender y razonar sobre el entorno de trabajo, y esto requiere de sistemas de percepción sensorial que tengan la capacidad el procesar los datos recopilados para transformarlos en información útil para entender el estado actual del entorno.

---

[5] http://wiki.ros.org/roslaunch
[6] http://wiki.ros.org/rviz
[7] http://gazebosim.org/
[8] http://wiki.ros.org/gazebo_ros_pkgs
[9] https://sir.upc.edu/projects/rostutorials/9-gazebo_sensors_tutorial/index.html
[10] https://sir.upc.edu/projects/rostutorials/10-gazebo_control_tutorial/index.html

En general, los sistemas de percepción tienen tres módulos esenciales, encargados de: (1) la adquisición y procesamiento de datos de sensores, (2) la representación de los datos o del modelo del entorno, y (3) el uso de algoritmos de aprendizaje automático e inteligencia artificial. En ROS se han incorporado utilidades y herramientas para cada uno de estos módulos dentro de un paquete llamado *ros-perception* con la finalidad de facilitar el desarrollo de sistemas de percepción que sean fácilmente reutilizables y escalables. La clave principal para ofrecer estas prestaciones ha sido la integración y compatibilidad de paquetes de trabajo (*toolboxes*) y librerías de terceros de código abierto. Esto es posible en gran medida gracias a la arquitectura modular de ROS que permite el uso de diferentes lenguajes de programación, tales como Java, C++ y Python.

ROS ofrece acceso a multitud de sensores de diferente tipo: sensores ópticos 1D/2D, cámaras 2D/3D, sensores de audio y reconocimiento de voz, sensores de fuerza/contacto, sensores de captura de movimiento, sensores de estimación de posición inerciales/GPS, entre otros. En el enlace oficial[11] se da un listado oficial de los sensores soportados en ROS. Los sistemas de percepción más desarrollados en ROS son aquellos basados en visión, tanto 2D y 3D. En este sentido, ROS ofrece dentro del paquete de percepción un conjunto de sub-paquetes tales como:

- IMAGE-COMMON: Contiene componentes para el transporte y compresión de imágenes, lectura/escritura de datos de calibración de cámaras en XML y guardar o recuperar información de la cámara.
- IMAGE-PIPELINE: Contiene componentes para procesar las imágenes que llegan desde la cámara de forma nativa (*raw*) tales como calibración, eliminación de distorsión, procesamiento de par estéreo, procesamiento de imágenes de profundidad, visualizador de imágenes, entre otras.
- IMAGE-TRANSPORT-PLUGINS: Contiene conectores (*plugins*) para el transporte de imágenes a través del sistema de subscrición/publicación de ROS con diferentes niveles de compresión y codificadores de video para reducir la latencia y el ancho de banda a través de la red.
- VISION-OPENCV: Contiene el conjunto de componentes que ofrecen una interfaz entre ROS y la librería OpenCV[12].
- PERCEPTION-PCL: Contiene el conjunto de componentes que ofrecen una interfaz entre ROS y la librería Point Cloud Library (PCL)[13].

Finalmente, cabe destacar que existen diferentes paquetes de ROS desarrollados sobre el paquete de *ros-perception* en los que se utilizan técnicas avanzadas y de reciente desarrollo, tales como modelos de inferencia basados en aprendizaje profundo generados en paquetes de terceros como Tensorflow[14] o PyTorch[15], o bien técnicas de aprendizaje por refuerzo gracias al uso de la librería OpenAI[16].

2.3.2. *Planificación de tareas*

Un robot con comportamiento inteligente debe tener capacidades de estimación de estado y de planificación y control de movimiento, y, además, poder combinarlas para llevar a cabo tareas de mayor complejidad. El campo de planificación automatizada de tareas (*AI Planning*) [17] lidia con este problema, buscando sintetizar automáticamente planes que combinan acciones básicas. Para ello, el planificador necesita dos elementos: un dominio y un problema. El dominio contiene un modelo de las posibles acciones y sus precondiciones y efectos, mientras que el problema contiene el estado actual del entorno y el objetivo final deseado. Estos se definen usando PDDL[18] (*Planning Domain Definition Language*), un lenguaje estándar para definir problemas de planificación automatizada.

[11] http://wiki.ros.org/Sensors#Portals
[12] https://opencv.org/
[13] https://pointclouds.org/
[14] https://www.tensorflow.org/
[15] https://pytorch.org/
[16] https://openai.com/
[17] https://planning.wiki/
[18] https://www.researchgate.net/publication/2278933_PDDL_-_The_Planning_Domain_Definition_Language

ROS ofrece una solución para desarrollar sistemas de planificación automatizada llamada ROSPlan[19] , que proporciona una colección de herramientas modulares para este propósito. El flujo de procesos en ROSPlan y los nodos que los implementan son los siguientes (los nodos se comunican a través de publicaciones, suscripciones y servicios predefinidos en el paquete de herramientas):

1. Base de datos (*Knowledge Base*): se actualiza con datos de los sensores y contiene los modelos PDDL (del dominio y del problema).
2. Interfaz del problema (*Problem Interface*): obtiene el dominio y el estado actual de la base de datos, y genera y publica el problema actual en PDDL.
3. Interfaz de planificación (*Planner Interface*): recibe el problema y el dominio PDDL y se lo pasa al planificador, para luego recibir y publicar el plan generado.
4. Interfaz de procesamiento (*Parsing Interface*): convierte el plan generado en acciones ejecutables que puedan ser enviadas a otros sistemas.
5. Envío del plan (*Plan Dispatch*): se encarga de conectar cada acción con proceso responsable de su ejecución, asegurándose de que las acciones se ejecuten correctamente.

Al ser un sistema formado por módulos, estos pueden ser modificados creando distintas arquitecturas y usando varios formalismos y lenguajes de planificación, incluyendo, por ejemplo, planificación determinística proposicional, temporal, contingente y probabilística.

2.3.3. Definición de comportamientos del robot mediante árboles de comportamiento

Una alternativa para la orquestación de las distintas habilidades básicas de un robot para que juntas compongan comportamientos complejos y adaptables es el uso de árboles de comportamiento (*Behavior Trees*). A diferencia de una máquina de estados finitos, un árbol de comportamiento es un árbol dirigido con nodos jerárquicos que controla el flujo de decisiones y la ejecución de "tareas" o "acciones". Existen tres categorías de nodos: el nodo raíz, los nodos controladores de flujo y los nodos de ejecución. La ejecución de un árbol de comportamiento comienza por el nodo raíz y continúa con los nodos siguientes de acuerdo con la estructura y lógica definida en el propio árbol. En cuanto a los nodos de control de flujo, existen nodos predefinidos que implementan diversas lógicas como, por ejemplo, los selectores y las secuencias:

- Los selectores buscan secuencialmente el primer nodo hijo en el árbol que devuelve un resultado de éxito (nótese que este hijo puede ser directamente un nodo de ejecución o una estructura de nodos que después de ejecutarse comunique su resultado). Estos nodos son muy relevantes al elaborar estrategias de actuación robustas a fallos porque permiten implementar un control con planes alternativos cuando el robot no puede ejecutar con éxito el plan predeterminado.
- Las secuencias ejecutan de forma sucesiva los nodos hijos que todavía no han finalizado su proceso (ya sea exitosamente o no). El valor de retorno de la secuencia será “fallo” o “en proceso” si cualquiera de los hijos devuelve “fallo” o “en proceso” respectivamente y sólo será de éxito una vez que todos los nodos hijos hayan devuelto un resultado de éxito.

Finalmente, los nodos de ejecución representan, conceptualmente, la funcionalidad más básica, y es donde se integran las capacidades primitivas del robot. Esta estrategia permite desacoplar el desarrollo de módulos, o componentes funcionales, del diseño y generación del comportamiento del robot.

ROS ofrece una herramienta intuitiva y versátil, *BehaviorTree.CPP* [20,21], para diseñar y ejecutar comportamientos del robot mediante árboles de comportamiento. Esta herramienta fue desarrollada originalmente por Eurecat en el contexto del proyecto Europeo RobMosys[22] y es mantenida actualmente (con código abierto) por la comunidad

[19] https://kcl-planning.github.io/ROSPlan/
[20] https://www.behaviortree.dev/
[21] https://github.com/BehaviorTree/BehaviorTree.CPP
[22] https://robmosys.eu/

organizada alrededor de ROS. El proyecto también incluye un editor gráfico llamado Groot[23] (ver Figura 1) específicamente pensado para el diseño intuitivo de comportamientos.

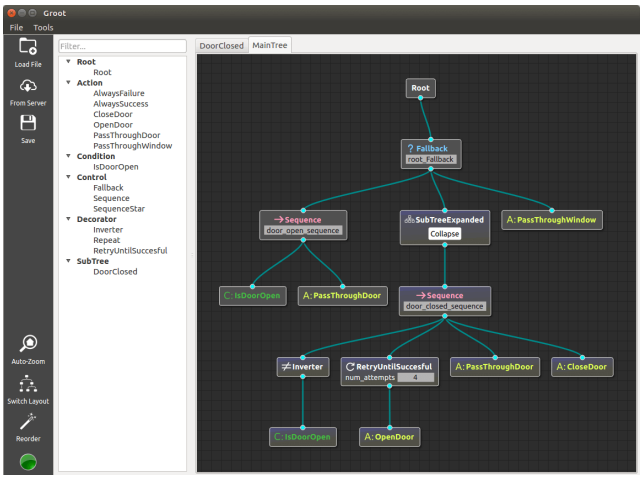


Figura 1. Ejemplo del editor gráfico Groot (fuente: nota al pie 23).

2.3.4. *Planificación del movimientos y trayectorias de robots manipuladores*

Cuando, por ejemplo, se quiere agarrar un objeto con un brazo robótico, se deben mover todas las articulaciones del robot para que la pinza pueda llegar al lugar adecuado para agarrar dicho objeto. Mover el brazo para lograr esa posición no es una tarea trivial, ya que se ha de generar la secuencia de posiciones que debe seguir cada articulación del brazo, en coordinación con las demás, de forma que la pinza se mueva desde su localización actual hasta la deseada. La determinación de esta secuencia coordinada de posiciones de las articulaciones se denomina planificación de movimientos, y MoveIt![24] es la herramienta de ROS más popular para hacerlo.

MoveIt! es compatible con más de 150 robots y trabaja con planificadores de movimientos a través de complementos (*plugins)*. Esto permite que MoveIt! se comunique y utilice planificadores de múltiples bibliotecas (por ejemplo, OMPL[25]), y por tanto sea fácilmente extensible. La interfaz con los diversos planificadores se realiza mediante una acción o servicio ROS (ofrecido por el nodo *move_group*). Los usuarios pueden acceder a las acciones y servicios proporcionados por *move_group* en C++, en Python y a través del complemento *Motion Planning* para Rviz (el visualizador de ROS mencionado en el Apartado 2.2). Además, *move_group* puede configurarse mediante el servidor de parámetros de ROS, del que también obtendrá el modelo URDF y el SRDF del robot (ver Figura 2).

El nodo *move_group* busca en el servidor de parámetros ROS otros datos específicos de MoveIt, incluyendo los límites de las articulaciones, la cinemática, el planificador de movimientos a usar, la interfaz de percepción, entre otros. Los archivos de configuración para estos datos son generados automáticamente por el asistente de configuración de MoveIt y almacenados en el directorio *config* del correspondiente paquete de configuración de MoveIt para el robot. *move_group* se comunica con el robot para obtener información de su estado actual (por ejemplo, las posiciones de las articulaciones), nubes de puntos y otros datos de los sensores y para dar instrucciones a los controladores del robot.

[23] https://github.com/BehaviorTree/Groot
[24] https://moveit.ros.org/
[25] https://ompl.kavrakilab.org/

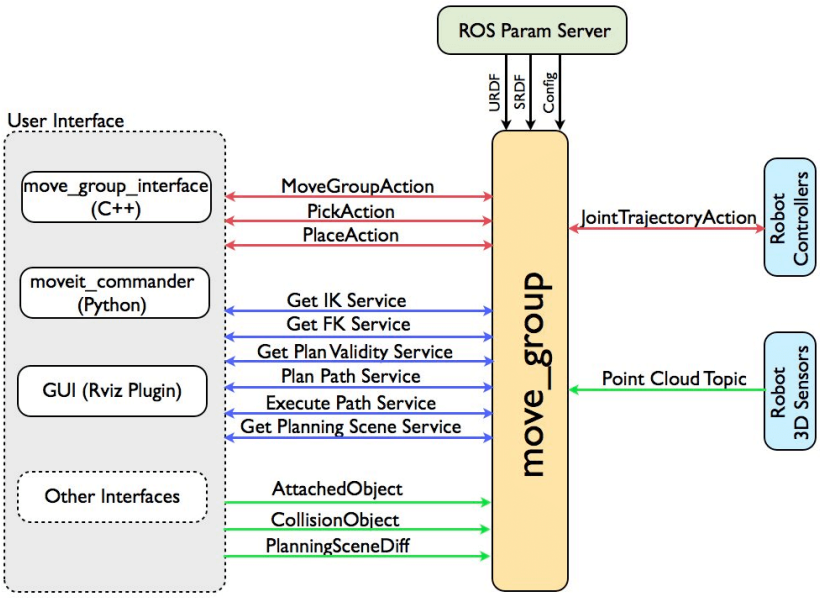


Figura 2. Mapa de parámetros, *topics*, acciones y servicios de ROS del nodo *move_group*.

Una vez configurado, el nodo *move_group* genera una trayectoria como respuesta a una solicitud de plan de movimientos. Por lo general, se le pide al planificador de movimientos que mueva un brazo a una nueva ubicación (en el espacio de las articulaciones) o el efector final a una nueva pose (en el espacio Cartesiano). Las colisiones se comprueban por defecto, incluyendo las de los objetos sujetos por la pinza y las auto-colisiones. También se pueden especificar restricciones adicionales para el planificador de movimientos como, por ejemplo, restricciones de posición, de orientación, de visibilidad, de las articulaciones u otras restricciones especificadas por el usuario. Una vez generada la trayectoria, el propio nodo *move_group* se comunica con los controladores del robot para ejecutarla.

2.3.5. *Localización y mapeado para robots móviles*

Uno de los mayores retos en robótica móvil es la estimación robusta del estado del robot (posición, orientación y velocidad) respecto a un sistema de referencia. La robótica probabilística busca distintas estrategias para fusionar múltiples sensores complementarios y reducir el error global. Por un lado, los sensores de medición relativa (odometría visual o de ruedas, giróscopos, acelerómetros, etc.) muestran errores inherentes que al ser integrados llevan a un error de estimación que crece con el tiempo. Por otro lado, los sensores de mediciones absolutas (GNSS, *compass*, etc.) no ofrecen el mismo nivel de consistencia y precisión local y pueden ser afectados por interferencias o incluso completamente ocluidos. El algoritmo clásico de estimación probabilística más utilizado en robótica se basa en el filtro de Kalman[26] y sus variantes cómo el filtro de Kalman extendido (EKF) o el "*unscented*" (UKF). ROS ofrece las dos implementaciones en el paquete Robot_Localization[27], junto con unas guías de configuración amigables y versátiles tanto para cualquier conjunto de sensores como para los parámetros del filtro. Una alternativa a los filtros de Kalman son los filtros de partículas, también llamados métodos Monte Carlo[28], que estiman el estado a partir del muestreo probabilístico de observaciones parciales. El paquete AMCL[29] es su principal aplicación en ROS, que utiliza un mapa 2D generado a partir de escáneres láser para localización global.

---

[26] Welch, G., & Bishop, G. (1995). An introduction to the Kalman filter.

[27] Moore, T., & Stouch, D. (2016). A generalized extended kalman filter implementation for the robot operating system. In *Intelligent autonomous systems 13* (pp. 335-348). Springer, Cham.

[28] Doucet, A., Godsill, S., & Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering. Statistics and computing, 10(3), 197-208.

[29] Thrun, Sebastian, et al. "Robust Monte Carlo localization for mobile robots." *Artificial intelligence* 128.1-2 (2001): 99-141.

Igualmente, un robot que se mueva autónomamente por el espacio de trabajo necesita una representación del mismo. El mapa del entorno puede tener distintos formatos (de rejilla de ocupación o de árbol topológico, entre otras) y dimensiones (2D o 3D) y su versión más sencilla se basa en la acumulación de los datos de los sensores de rango proyectados sobre una fuente de localización que ha de ser fiable. Cualquier error tanto en la localización como en los datos del sensor lleva a incoherencias, distorsiones u otros errores en el mapa. Para resolver este problema, el algoritmo de localización y mapeado simultáneo (SLAM) permite que el robot sea capaz de reconocer características y referencias previamente enseñadas y, mediante técnicas de optimización, corregir la estimación de la posición y rectificar los errores y distorsiones del mapa. ROS ofrece múltiples implementaciones de un sistema de SLAM en 2D para robots terrestres como, por ejemplo, GMAPPING[30], un método basado en filtro de partículas que genera un mapa de ocupación 2D a partir de escáneres láser y odometría, Karto SLAM[31], que utiliza optimización de grafos para generar el mapa y estimar la posición, y CARTOGRAPHER[32], también basado en grafos y que permite la actualización y utilización de mapas solo para localización. De igual forma, existen en ROS implementaciones de SLAM en 3D basadas en LIDAR y cámaras, como ORB-SLAM[33], y RTAB-MAP[34].

2.3.6. *Planificación de trayectorias y navegación para robots móviles*

En ROS, el módulo de navegación autónoma, denominado *Navigation Stack*[35] (*NavStack*), es el componente software responsable de la generación de las trayectorias libres de colisiones con el entorno para robots móviles, así como de su traducción a los comandos de control del robot. Para ello, usa un mapa, generado anteriormente por el módulo de localización y mapeado, juntamente con información de sensores como láseres o cámaras. Básicamente, utiliza como información de entrada el mapa, la localización, la odometría del robot y las medidas proporcionadas por los sensores instalados en la plataforma y produce como resultado comandos de velocidad que son enviados a la base móvil.

La Figura 3 muestra el diagrama de nodos y datos que constituyen la arquitectura de navegación *NavStack*. El nodo principal de *NavStack* se denomina *move_base* (centro de la imagen). Los nodos de color blanco constituyen elementos obligatorios para el funcionamiento de *NavStack* que ya están implementados en ROS. Los nodos de color gris son opcionales y también ya están disponibles en ROS. Por último, los nodos azules deben crearse y/o configurarse explícitamente para cada plataforma robótica con la que se desee trabajar (existen multitud de ejemplos para los sensores y plataformas más comunes).

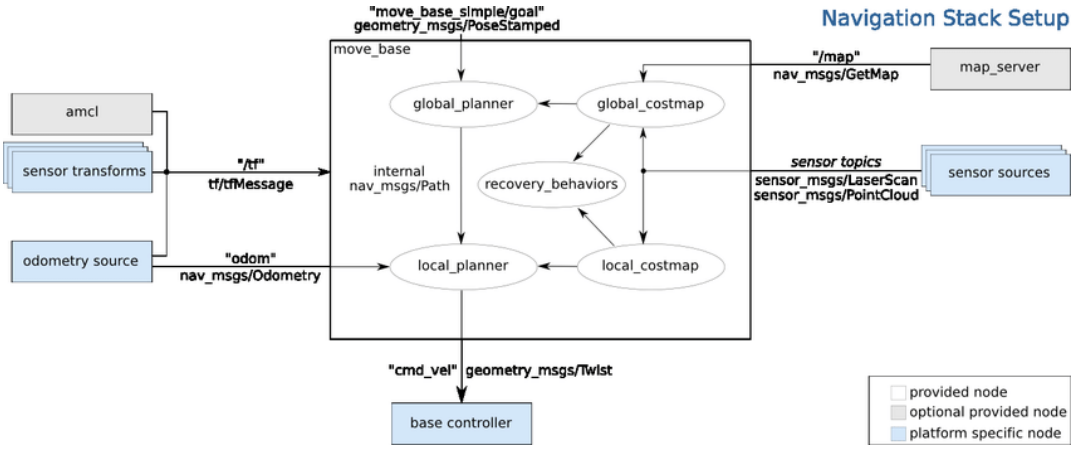


Figura 3. Arquitectura de navegación *NavStack* implementada en el framework robótico ROS (fuente: nota al pie 35).

[30] http://wiki.ros.org/gmapping.
[31] http://wiki.ros.org/slam_karto.
[32] https://github.com/cartographer-project/cartographer.
[33] https://github.com/appliedAI-Initiative/orb_slam_2_ros.
[34] http://wiki.ros.org/rtabmap_ros
[35] http://wiki.ros.org/navigation

Los componentes más relevantes son:

- Mapas de costes (*costmaps)*, local y global: una vez que el módulo de SLAM construye un mapa, la implementación en ROS del módulo de navegación lo convierte en un mapa de costes, que es una estructura dinámica de datos que mantiene, de manera eficiente, información sobre las zonas del entorno por las que el robot puede navegar de manera segura sin colisionar con otros objetos.
- Planificador global (*global planner*), es el responsable de la generación de un camino de navegación global sobre el espacio libre. Tiene como entrada un mapa (generalmente un mapa de rejilla de ocupación en forma de *costmap*), y, considerando la forma y restricciones cinemáticas del robot genera un camino para la navegación. En ROS existen implementaciones de algoritmos como el A*[36], Dijskstra[37], Voronoi[38] o SBPL[39]. Hay que destacar que en el mapa tomado como entrada aparece la estructura del entorno y aquellos objetos (generalmente estáticos) que se encontraban en el rango de visión de los sensores láser en el momento de generar el mapa.
- Planificador local (*local planner*): en caso de detección de obstáculos no representados en el mapa de entrada, que por tanto no toma en cuenta el planificador global, la función del planificador local es la de sortear estos obstáculos para posteriormente converger al camino de navegación global, evitando así colisiones. Este módulo tiene como entrada información directa de los sensores, como láseres o cámaras. En ROS, existen implementaciones de algoritmos de planificación local como, por ejemplo, la ventana dinámica (Dynamic Window Approach)[40], Bandas Elásticas[41], Timed-Elastic bands[42], entre otros.

2.4. ROS industrial

Tal como ya se mencionó anteriormente, ROS ha sido durante muchos años el sistema más utilizado para la programación de robots en entornos académicos y de investigación. La iniciativa ROS-Industrial nace con el objetivo de aprovechar los desarrollos realizados en ROS, y trasladarlos al mundo de la robótica industrial y la automatización. Para ello, ROS-Industrial se constituye como un proyecto de código abierto, traccionado y financiado por un consorcio de empresas de índole principalmente industrial, incluyendo empresas del mundo de la robótica industrial, como ABB, Omron, Universal Robots o Yaskawa, usuarios finales con gran interés en la automatización avanzada, como Airbus, Boeing o 3M, empresas interesadas en el uso de ROS para proyectos de robótica avanzada o incluso conducción autónoma, como Bosch, BMW o Volvo, y grandes empresas con intereses en la Inteligencia Artificial y su relación con la robótica, como Amazon Web Services, Microsoft o Intel.

Es importante destacar que ROS-Industrial no es un sistema que compita o que sea una alternativa a ROS. A nivel de sistema, ROS-Industrial utiliza ROS como su base, sobre la que añade funcionalidades y paquetes. Por tanto, ROS-Industrial puede entenderse en muchos aspectos como desarrollos adicionales sobre el propio ROS, además de un compendio de actividades de soporte.

---

[36] D. Gelperin, "On the optimality of A*", Artificial Intelligence, vol. 8, no. 1, pp. 69–76, 1977.

[37] S. Skiena, "Dijkstra's algorithm", in Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, pp. 225–227, Addison-Wesley, Boston, Mass, USA, 1990.

[38] Garber M., Lin M.C. Constraint-Based Motion Planning Using Voronoi Diagrams. In: Boissonnat J.D., Burdick J., Goldberg K., Hutchinson S., editors. Algorithmic Foundations of Robotics V. Springer; Berlin/Heidelberg, Germany: 2004. pp. 541–558

[39] B. J. Cohen, S. Chitta and M. Likhachev . "Search-based planning for manipulation with motion primitives," 2010 IEEE International Conference on Robotics and Automation, 2010, pp. 2902-2908.

[40] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine. 4 (1): 23–33, 1997.

[41] Quinlan, S. and Khatib, O. Elastic Bands: Connecting Path Planning and Robot Control. Proc. IEEE International Conference on Robotics and Automation, Atlanta, Georgia 1993, vol 2. pp. 802-807.

[42] Rösmann C., Feiten W., Wösch T., Hoffmann F. and Bertram T. Efficient trajectory optimization using a sparse model. Proc. IEEE European Conference on Mobile Robots, Spain, Barcelona, 2013, pp. 138–143.

De acuerdo a la propia visión de ROS-Industrial, el consorcio trabaja sobre 3 ejes fundamentales:

1. Identificar los requerimientos de robótica industrial y automatización, y trabajar para llevarlos al ecosistema ROS.
2. Trasladar e institucionalizar conceptos de calidad de código fuente en ROS.
3. Actividades de soporte, formación y diseminación.

En cuanto al primer punto, uno de los resultados más visibles es el de la definición de las interfaces de distintos tipos de equipamiento industrial, principalmente de las interfaces para la comunicación con los controladores robóticos industriales, ya que cada fabricante expone de forma distinta las capacidades de sus controladores, e incluso distintos robots ofrecen distintas capacidades. De cara a unificar la comunicación desde un sistema ROS, el consorcio ROS-Industrial ha definido las capacidades mínimas y las interfaces[43] que debe implementar el controlador (entendido aquí como controlador software o *driver*) para garantizar la compatibilidad. En esta línea, el consorcio de ROS-Industrial también ha desarrollado los controladores software necesarios para comunicar cualquier programa ROS con robots de distintos fabricantes, tales como ABB[44], Universal Robots[45] o Fanuc[46].

Respecto al segundo punto, la calidad del código fuente es a menudo una preocupación cuando se trata de un proyecto de código abierto, donde hay contribuidores de distintos perfiles, desde profesionales con larga experiencia hasta estudiantes. Para conseguir la confianza de los actores industriales, ROS-Industrial trabaja en este aspecto, desarrollando un sistema que permite de forma sencilla ejecutar tareas de integración continua (*continuous integration*[47]) sobre paquetes ROS, desarrollando diversas herramientas que permiten obtener medidas objetivas de la calidad del código fuente[48], y estableciendo una clasificación que permite a un desarrollador conocer el grado de madurez de un componente ROS antes de descargarlo[49]. Esta clasificación, que se aplica a los paquetes soportados por ROS-Industrial, clasifica cada paquete como *experimental, en desarrollo o en producción*.

Por último, en relación al tercer punto, todo gran proyecto de código abierto necesita de actividades que fomenten su uso por parte de terceros. Para ello, el consorcio de ROS-Industrial ofrece tareas de soporte[50] y formación[51], genera material específico para la formación[52] y celebra distintos eventos en torno a ROS como conferencias anuales[53].

2.5. ROS2

¿Debería utilizar ROS o ROS2 para mi proyecto robótico? ¿Está listo ROS2 para su uso en aplicaciones en producción? ¿Es ROS2 un sistema completamente nuevo? ¿Son compatibles mis desarrollos anteriores en ROS sobre el nuevo ROS2? Estas son algunas de las preguntas más comunes que se hacen algunos recién llegados al mundo ROS y a las que daremos respuesta en esta sección.

2.5.1. *El por qué de ROS2*

Cuando inicialmente se creó ROS en el año 2007, uno de los preceptos principales que se tuvo en cuenta fue el de crear un sistema que permitiese al máximo posible la reutilización de código, dando soporte a distintos tipos de

[43] http://wiki.ros.org/Industrial/Industrial_Robot_Driver_Spec
[44] https://github.com/ros-industrial/abb
[45] https://github.com/UniversalRobots/Universal_Robots_ROS_Driver
[46] https://github.com/ros-industrial/fanuc
[47] https://github.com/ros-industrial/industrial_ci
[48] http://wiki.ros.org/code_quality
[49] http://wiki.ros.org/Industrial/Software_Status
[50] https://rosindustrial.org/tech-support
[51] https://rosindustrial.org/events/2021/3/9/ros-2-industrial-training-kggfk-er39t-rdyfp
[52] https://github.com/ros-industrial/ros2_i_training
[53] https://rosindustrial.org/events/2021/12/1/ros-industrial-conference-2021

robots y de aplicaciones. Sin embargo, su diseño y desarrollo se produjo inicialmente dentro de un contexto muy específico, y su adopción en el ámbito industrial ha introducido nuevos requerimientos, algunos de los cuales se han podido implementar en ROS mediante de parches, como *multimaster_fkie*[54] para soporte *multimaster* o SROS[55] para añadir mecanismos de seguridad en las comunicaciones. Hacia el año 2014, para poder implementar más adecuadamente estos nuevos requerimientos, y, sobre todo, dar un mejor soporte al mundo industrial y sus necesidades, se decide dar un salto y comenzar la implementación de ROS2. Dos aspectos principales de este salto son:

- ROS2 no es un sistema completamente nuevo, sino que nace y se apoya en ROS; muchos de los conceptos de ROS siguen presentes en ROS2 (nodos, tópicos, servicios, etc) y muchas de las aplicaciones en las que se apoya ROS2 han sido portadas desde ROS (*RViz* para la visualización o *MoveIt!* como sistema de planificación, entre otras).
- ROS2 no es una nueva versión de ROS, y por tanto no mantiene la compatibilidad "hacia atrás"; eso significa que en general, el código desarrollado en ROS no funcionará en ROS2.

Para facilitar la transición de los desarrolladores a ROS2 se proporcionan una serie de recursos, tales como:

- Guía de migración de ROS[56]: un documento que detalla los cambios principales a realizar a código desarrollado sobre ROS para migrarlo a ROS2.
- *ros1_bridge* [57] : un nodo que actúa como intermediario de mensajes, permitiendo que trabajen conjuntamente nodos escritos en ROS y ROS2 en un mismo sistema.
- *rospy2* [58] : una librería Python que ofrece una interfaz igual a la librería *rospy* de ROS, pero que internamente utiliza ROS2, lo que permite que, con un mínimo cambio, un nodo Python escrito para ROS pueda funcionar en ROS2.

2.5.2. *Principales diferencias entre ROS y ROS2*

El cambio que ha impulsado la implementación de ROS2 ha sido la adopción de DDS (Data Distribution Service) como capa de comunicaciones. DDS es un estándar definido por el OMG (Object Management Group), un consorcio dedicado al establecimiento de estándares en el ámbito de las tecnologías industriales. DDS se utiliza de forma extensiva en la industria y tiene un largo historial de uso en aplicaciones críticas, tales como sistemas de vuelo y sistemas financieros, lo que, además de otros motivos técnicos, hacen de DDS un buen sistema sobre el que cimentar el enfoque industrial de ROS2. Al tratarse de una especificación, existen distintas implementaciones de DDS disponibles, tanto de forma comercial como de código abierto. Cada una de estas implementaciones ofrece distintas características en cuanto a, por ejemplo, plataformas soportadas, rendimiento, o licencias. Para permitir al desarrollador seleccionar la implementación más adecuada, ROS2 define su propia interfaz sobre DDS, lo que le permite soportar distintas implementaciones, siendo Fast DDS y Cyclone DDS las dos con más implantación en el mundo ROS2.

Además de la adopción de DDS, ROS2 introduce varios cambios relevantes con el objetivo de solucionar algunas de las limitaciones de ROS y de entrar en el mundo de las aplicaciones industriales. En la Tabla 1 se resumen algunos cambios más significativos que ha introducido ROS2 en comparación con las limitaciones identificadas en ROS.

[54] http://wiki.ros.org/multimaster_fkie
[55] http://wiki.ros.org/SROS
[56] https://docs.ros.org/en/rolling/Contributing/Migration-Guide.html
[57] https://github.com/ros2/ros1_bridge
[58] https://github.com/dheera/rospy2/

Tabla 1. Cambios más significativos entre ROS y ROS2.

ROS	ROS2
Capa de comunicación basada en un *middleware* ad-hoc y TCP como capa de transporte.	Capa de comunicación basada en el estándar DDS, soportando distintas implementaciones del mismo.
Comunicación centralizada, ya que el *Master* es responsable de establecer las comunicaciones, lo que lo convierte en un posible punto único de fallo.	DDS permite tener un sistema completamente descentralizado y que el descubrimiento de nodos se realice de forma distribuida.
Sin posibilidad de establecer Calidad del Servicio (QoS, del inglés *Quality of Service*), siendo lo más cercano los *Transport Hints*[59]	Apoyándose en DDS, ofrece distintas configuraciones de QoS; además es posible tener un mayor control modificando la configuración de la propia implementación de DDS.
Sin soporte multi-robot, principalmente debido a la limitación de la existencia de un único *Master*, lo que dificulta la gestión de múltiples sistemas robóticos que trabajen en redes independientes.	La capa de comunicación basada en DDS permite un sistema totalmente distribuido, permitiendo gestionar múltiples redes.
Oficialmente sólo es soportado en Ubuntu, aunque en las últimas versiones también hay paquetes binarios para Windows[60].	Multiplataforma, el sistema de integración continua que se encarga de verificar y generar los paquetes binarios de ROS2 soporta Ubuntu, Windows y Mac OS X. Además, gracias al uso de DDS-XRCE (DDS for Extremely Resource Constrained Environments) también es posible utilizar micro-ROS[61] para utilizar ROS2 sobre microcontroladores.
Sin soporte para tiempo real. En caso de requerir tiempo real, es habitual el uso de librerías externas como Orocos[62].	Aunque no garantiza la ejecución en tiempo real, su arquitectura hace más sencillo el desarrollo de componentes que funcionen en tiempo real. El uso de DDS para las comunicaciones (que soporta tiempo real), las posibilidades de uso en distintas plataformas (por ejemplo, microcontroladores), y el *Executor*[63] introducido para gestionar la ejecución facilita a los desarrolladores el conseguir un comportamiento determinista.
Comunicaciones no seguras, aunque algunos sistemas como SROS[64] o algunos paquetes como rosauth[65] introducen mecanismos de seguridad.	Utiliza 3 mecanismos de seguridad de la especificación de DDS-*Security Specification*[66]: • Autenticación • Control de acceso • Criptografía y encriptación

[59] http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers#Transport_Hints
[60] http://wiki.ros.org/Installation/Windows
[61] https://micro.ros.org/
[62] https://orocos.org/
[63] https://docs.ros.org/en/galactic/Concepts/About-Executors.html
[64] http://wiki.ros.org/SROS
[65] http://wiki.ros.org/rosauth
[66] https://www.omg.org/spec/DDS-SECURITY/1.1/PDF

3. Recursos de aprendizaje y foros relacionados

Existe en Internet abundante documentación sobre ROS. Se detalla a continuación una recopilación de enlaces que se consideran significativos como recursos de aprendizaje.

3.1. Documentación ROS

El punto de partida es la página principal de ROS[67], con información general de qué es ROS, la wiki[68], como portal de entrada a la documentación de ROS 1, y la página principal de la documentación de ROS 2 en su versión actual Galactic[69]. También son interesantes el sitio de preguntas y respuestas[70] y el de foro de discusión[71].

3.2. Cursos

En la wiki de ROS se provee de enlaces[72] a varios cursos cortos y a cursos impartidos por distintas universidades. Los cursos cortos son cursos de pocos días, la gran mayoría de pago, que cubren un abanico amplio de temas, desde los básicos a algunos más avanzados. Cabe destacar los cursos ofrecidos por *The Construct*[73] (que no requieren de la instalación de ROS), *Robocademy*[74] (que incluye un blog con una entrada referente a recursos gratuitos) *y Udemy*[75]. De los cursos impartidos por universidades se da básicamente información del syllabus, aunque cabe destacar que en dos de ellos (ETSEIB-UPC BarcelonaTech y ETH Zurich) se incluye el enlace al material de aprendizaje completo (tutoriales[76] / videos[77]).

3.3. Tutoriales, libros y proyectos de aprendizaje.

En la wiki de ROS se provee de enlaces a varios tutoriales de aprendizaje[78], que incluyen temas nucleares de ROS (como paquetes, nodos, comunicaciones con *topics* y *services*, y *launch*, entre otros), así como de modelado[79], visualización[80], navegación[81], comunicaciones con *actionlib*[82], ROS-Industrial[83], *tf*[84], o el uso de PCL con ROS[85].

---

[67] https://www.ros.org/
[68] http://wiki.ros.org/
[69] http://docs.ros.org/en/galactic/
[70] https://answers.ros.org/
[71] https://discourse.ros.org/
[72] http://wiki.ros.org/Courses/
[73] https://www.theconstructsim.com/
[74] https://robocademy.com/
[75] http://www.riotu-lab.org/udemy.php
[76] https://sir.upc.edu/projects/rostutorials
[77] https://youtube.com/playlist?list=PLE-BQwvVGf8HOvwXPgtDfWoxd4Cc6ghiP
[78] http://wiki.ros.org/ROS/Tutorials
[79] http://wiki.ros.org/urdf/Tutorials
[80] http://wiki.ros.org/visualization/Tutorials
[81] http://wiki.ros.org/navigation/Tutorials
[82] http://wiki.ros.org/actionlib_tutorials/Tutorials
[83] http://wiki.ros.org/Industrial/Tutorials
[84] http://wiki.ros.org/tf/Tutorials
[85] http://wiki.ros.org/pcl/Tutorials

Asimismo la wiki provee enlaces a libros[86], de entre los que cabría destacar "A Gentle Introduction to ROS"[87], disponible *online*, y "Mastering ROS for Robotics Programming"[88].

Finalmente, se debe destacar que la difusión de ROS, en su vertiente ROS-Industrial, se ha promovido en la Unión Europea mediante el proyecto ROSIn[89], que incluyó una vertiente de educación con cursos MOOC (Massive Open Online Courses) [90].

4. Adaptación de dispositivos a ROS

Uno de los aspectos de mayor interés dentro de la comunidad ROS, y por el que muchos usuarios optan por este sistema, es la facilidad en la integración de diferente hardware en los sistemas robóticos gracias a la disponibilidad de nodos ROS o *drivers* ya desarrollados para toda clase de dispositivos involucrados en estos sistemas, desde los brazos robóticos propiamente dichos, hasta cámaras o sistemas LiDar, entre muchos otros. ROS mantiene listas de todos los dispositivos que lo soportan: robots manipuladores[91], robots móviles[92], y pinzas, manos mecánicas y otros componentes[93], de entre los que destacan aquellos que de forma nativa están desarrollados para ROS como los de Clearpath Robotics, Fetch Robotics, Pal Robotics, Robotnik o Shadow Robot.

La comunidad ROS continuamente aporta nuevos *drivers* y provee una serie de directrices[94] para desarrollar código con un alto grado de calidad. No resulta descabellado afirmar que es relativamente sencillo encontrar *drivers* ya existentes para un componente hardware determinado, y también que normalmente éstos permiten aplicar toda su funcionalidad y disponer de los datos asociados prácticamente de forma inmediata dentro de un entorno o sistema basado en ROS. No obstante, no siempre se pueden garantizar unos *drivers* robustos, desarrollados en base a estándares o normas, criterios de calidad de software, y con un nivel de test e integración debidamente realizados, y, además, bien documentados.

Si bien estos requisitos de calidad podrían ser más menos rigurosos en algunos desarrollos en el ámbito académico y/o de investigación aplicada, para su integración en sistemas y productos comerciales, tanto en robótica de servicio como robótica industrial, sí que suponen un aspecto muy relevante para los Integradores y Fabricantes de Equipamiento (OEM, del inglés *Original Equipment Manufacturer*), los cuales requieren de altos niveles de fiabilidad y soporte técnico.

Un ejemplo de paquete desarrollado en cumplimiento de estos requisitos es el *psen_scan_v2*[95] desarrollado por la empresa PILZ para su escáner de seguridad PSENscan[96]. Este paquete de código abierto, además de estar desarrollado por el propio vendedor del equipo, ha sido probado a diferentes niveles en cada una de las versiones liberadas, concretamente se han llevado a cabo pruebas a nivel de código (*unit test* e *integration test*) así como pruebas incluyendo el propio hardware (*acceptance test*), todas ellas descritas y documentadas al alcance de la comunidad (ver Figura 4). Por otro lado, el paquete también está extensamente documentado, tanto a nivel de usuario (guía de uso, manual, tutoriales) como de desarrollador (diagrama y documentación de clases y diagramas de operación[97] mediante Lenguaje Unificado de Modelado -UML, del inglés Unified Modeling Language-). Además, el propio fabricante se ocupa del mantenimiento del paquete y provee un correo de soporte gratuito.

---

[86] http://wiki.ros.org/Books
[87] https://www.cse.sc.edu/~jokane/agitr/
[88] https://www.packtpub.com/product/mastering-ros-for-robotics-programming-third-edition/9781801071024
[89] https://www.rosin-project.eu/
[90] https://online-learning.tudelft.nl/courses/hello-real-world-with-ros-robot-operating-systems/
[91] https://robots.ros.org/category/manipulator/
[92] https://robots.ros.org/category/ground/
[93] https://robots.ros.org/category/component/
[94] http://wiki.ros.org/Quality
[95] https://github.com/PilzDE/psen_scan_v2
[96] https://www.pilz.com/en-INT/products/sensor-technology/optoelectronic-protective-devices/psenopt
[97] https://github.com/PilzDE/psen_scan_v2/tree/main/doc

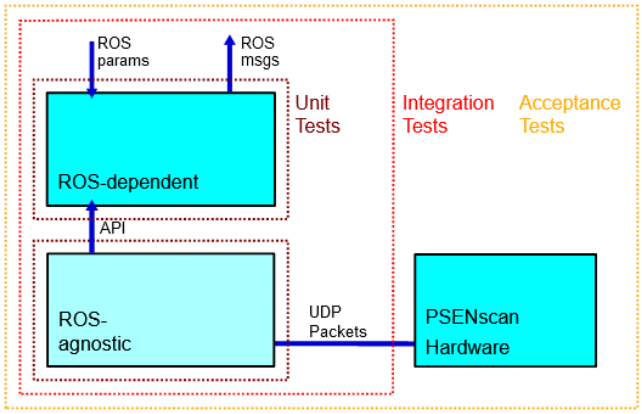


Figura 4. Descripción de la estructura de los test realizados al código psen_scan_v2, incluyendo Unit Test de diferentes componentes, Test de integración y Test de aceptación incluyendo el hardware.

5. Casos de estudio (aplicaciones en las que se usa ROS)

5.1. Arquitectura de control genérica basada en ROS para un desarrollo más rápido y simplificado de nuevas aplicaciones robóticas

Las capacidades de los sistemas robóticos se han ampliado al fusionarse los manipuladores fijos con las plataformas móviles teleoperadas, dando lugar a los robots manipuladores móviles, lo que permite al robot desplazarse al lugar de interés para realizar la tarea deseada. Esta fusión ha dado lugar a diferentes soluciones ad-hoc que normalmente son complejas y no son reutilizables al cambiar el entorno. En este contexto, Tekniker ha creado *Robotframework*, una arquitectura genérica basada en ROS que permite integrar fácilmente diferentes tipos de navegación, manipulación, percepción y módulos de alto nivel que facilitan un desarrollo más rápido y simplificado de nuevas aplicaciones robóticas. La arquitectura incluye herramientas genéricas de recogida de datos en tiempo real, módulos de diagnóstico y gestión de errores, así como interfaces de usuario fáciles de usar.

La arquitectura, de cuatro capas, busca la fácil integración de las diferentes funcionalidades del robot mediante un diseño de computación distribuida que permite que varias tareas se ejecuten en diferentes ordenadores sin dejar de aparecer ante sus usuarios como un único sistema coherente, permitiendo además una fácil extensibilidad. En la arquitectura hay varios componentes comunes a cualquier aplicación, representados en color azul en la Figura 5, que incluyen la interfaz de usuario del robot, toda como la capa de aplicación, los módulos de gestión de errores y algunas partes comunes de la capa de habilidades.

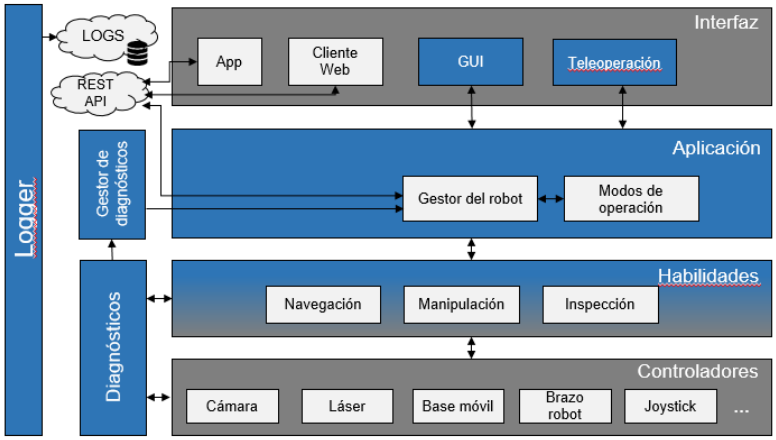


Figura 5. Arquitectura de control de cuatro capas de Robotframework. Los módulos representados en azul son comunes a cualquier aplicación.

La capa de *drivers* incluye los módulos que permiten interactuar con los sensores y actuadores de la plataforma robótica. Una de las ventajas de utilizar ROS es la disponibilidad de una amplia variedad de controladores de diferentes elementos, como la base móvil, el manipulador, las cámaras y los sensores láser. Esto hace que la arquitectura sea genérica desde el punto de vista del hardware, permitiendo la posibilidad de sustituir elementos sin modificar la arquitectura.

La capa de habilidades está compuesta por los nodos de ROS que participan en las funcionalidades básicas de control de un robot. Estos nodos gestionan los sensores y actuadores, y proporcionan capacidades del robot como la navegación, la manipulación y la inspección autónomas. En este nivel, ROS proporciona una amplia gama de paquetes con funciones específicas: *GMapping* para generar mapas; el *stack* de navegación para planificar trayectorias globales y locales que permitan a la base móvil moverse evitando obstáculos; la descripción unificada del robot en formato URDF (del inglés *Unified Robot Description Format*); *MoveIt!* para generar y ejecutar trayectorias de manipulación libre de colisiones. La arquitectura incluye módulos adicionales para la navegación relativa y precisa basada en marcas, que son valiosos para una amplia gama de aplicaciones de robóts móviles.

La capa de aplicación incluye el módulo de gestión del robot, que se encarga de controlar todo el sistema robótico, y el módulo con el modo de operación, que interpreta los planes de alto nivel para una aplicación robótica específica. El modo de operación está diseñado para ser lo más general posible y fácilmente configurable para una variedad de procesos robotizados, y, por lo tanto, para evitar implementaciones ad-hoc útiles sólo para configuraciones específicas del espacio de trabajo. Un plan determina un conjunto de objetivos que fijan los destinos para la navegación la plataforma móvil y un conjunto de tareas a realizar en cada destino. Los planes se especifican en ficheros con formato JSON (del inglés *JavaScript Object Notation*), un formato estándar abierto muy común e independiente del lenguaje usado, que utiliza texto legible por el ser humano para almacenar y transmitir datos.

La capa de interfaz incluye los módulos relacionados con la interacción entre el usuario y el sistema, como son el módulo con la interfaz gráfica de usuario y el módulo de teleoperación. El módulo con la interfaz gráfica es una aplicación de escritorio que ofrece al usuario la capacidad de iniciar o detener los modos de operación y ver el estado del sistema, incluyendo cualquier alerta de estado. El módulo de teleoperación permite el control manual del sistema robótico mediante el uso de un joystick. El sistema ofrece también un API de REST[98] que permite la interacción con otros servicios, como aplicaciones web o móvil.

Los tres módulos mostrados en la parte izquierda de la Figura 5 están destinados a monitorizar el estado funcional del sistema. El módulo de diagnóstico ha sido diseñado para recoger y preprocesar datos específicos de los

[98] https://www.redhat.com/es/topics/api/what-is-a-rest-api

controladores y de la capa de habilidades que luego se pasan al módulo gestor de diagnóstico para la toma de decisiones automática y la notificación de incidencias. Los *logs* se utilizan para registrar el seguimiento histórico del proceso, y pueden almacenarse tanto localmente en el robot como en la nube, utilizando una base de datos no relacional.

Robotframework se ha utilizado para el desarrollo de un sistema robótico capaz de realizar una monitorización autónoma y continua en invernaderos que permite la detección, identificación y control de pestes (en el contexto del proyecto europeo GreenPatrol[99]), y también para el desarrollo de una solución de robótica colaborativa para la inspección mediante ultrasonidos de componentes aeronáuticos (en el contexto del proyecto Cro-Inspect[100]). La Figura 6 muestra imágenes de ambas aplicaciones, en las que se ha podido constatar que el uso de *Robotframework* puede reducir significativamente el tiempo de preparación de nuevas aplicaciones robóticas con manipuladores móviles en tareas relacionadas con la agricultura y la industria.

a) b)

Figura 6. Aplicaciones en que se ha utilizado Robotframework: a) Plataforma robótica Greenpatrol, y b) Inspección avanzada de piezas compuestas complejas.

5.2. Célula robótica para acabado superficial de piezas

Ciertas operaciones de procesado superficial de piezas tales como pulido o rebarbado son labores tediosas, cuya automatización suele ser compleja por lo que habitualmente se realizan de forma manual. Para los casos de rebarbado de piezas por ejemplo, se estima que un sistema robótico que combine las capacidades de visión artificial y un sistema ágil de programación de las operaciones puede suponer un ahorro de tiempo de en torno al 60%.

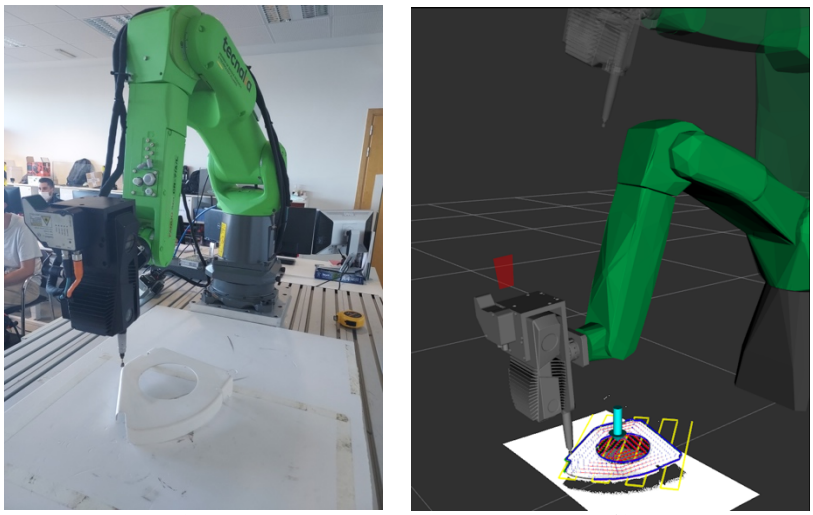
Las empresas del sector de la fundición utilizan principalmente procesos de rebarbados manuales o semi-manuales. Sin embargo, son aplicaciones susceptibles de ser robotizadas, dadas las ventajas que presenta frente a su ejecución manual: resultados no homogéneos, trabajo manual penoso, variabilidad en los tiempos y calidades, etc.

La tecnología ScanNPlan es un buen ejemplo de la construcción de una aplicación robótica compleja a partir de las distintas funcionalidades y módulos de los que provee ROS. Se trata de un conjunto de herramientas que permiten la generación de trayectorias en tiempo real, basada en un escaneo 3D. El enfoque ScanNPlan supera las limitaciones en la programación tradicional de robots para aplicaciones que:

- Tienen alta variabilidad de piezas, haciendo ineficiente la programación manual.
- No tienen modelización CAD de las piezas.
- Incluyen piezas flexibles o deformables, haciendo imposible la programación fuera de línea.
- Requieren fijación de piezas flexible o ninguna fijación.

[99] https://cordis.europa.eu/project/id/776324/es
[100] https://cordis.europa.eu/project/id/716935/es

Basado en el sistema ScanNPlan, TECNALIA ha desarrollado una célula robótica capaz de realizar procesos de pulido y rebarbado sobre piezas de distinta geometría. La célula consta principalmente de un robot colaborativo Fanuc CR-7iAL, un sistema de visión 3D de luz blanca estructurada, un perfilómetro láser y una herramienta seleccionada en función del proceso a realizar (Figura 7). Gracias al sistema modular de ROS y el soporte a distinto hardware (tanto los controladores de distintos robots como sensores), es posible intercambiar el equipamiento en función a los requerimientos de la aplicación sin que tenga un gran impacto sobre el grueso del desarrollo de la aplicación.



a) b)

Figura 7. Detalle de la célula robótica: a) real, y, b) simulada en RViz.

Los bloques principales de la aplicación son:

- Escaneo de la pieza: dada la incertidumbre en la posición y tamaño de la pieza, se realizan varias capturas con el sensor de visión 3D en el espacio de trabajo, que se registran formando una única nube de puntos. Hay distintos paquetes de ROS para realizar la calibración de sensores, tanto la calibración intrínseca[101] como la extrínseca[102].
- Tratamiento y extracción de la nube de puntos: a partir de la nube de puntos obtenida en el escaneo, se usa la integración de la librería PCL[103] en ROS para filtrar y tratar la nube de puntos hasta obtener un mallado que será utilizado por los algoritmos de planificación automática.
- Generación automática de trayectorias: el planificador Descartes[104] permite generar trayectorias en el espacio Cartesiano, donde además puede haber grados de libertad no restringidos (Figura 8). Este es el caso de muchas aplicaciones industriales donde, por ejemplo, la orientación de la herramienta no es relevante.
- Simulación e interacción con el operario: las trayectorias generadas anteriormente se simulan para mostrar el resultado al operador. En este caso no es necesario utilizar un simulador complejo con motor físico como puede ser Gazebo; el paquete *industrial_robot_simulator*[105] permite simular el controlador robótico y ejecutar las trayectorias generadas. El operario tiene una interfaz gráfica sencilla donde puede cambiar algunos parámetros (por ejemplo, la distancia entre cada pasada) y ver su efecto en una nueva simulación.

[101] http://wiki.ros.org/camera_calibration
[102] https://github.com/ros-industrial/industrial_calibration
[103] https://pointclouds.org/
[104] https://github.com/ros-industrial-consortium/descartes
[105] https://github.com/ros-industrial/industrial_core/tree/melodic-devel/industrial_robot_simulator

- Ejecución de las trayectorias: la especificación de la comunicación de ROS-Industrial[106] define cómo enviar mensajes a un controlador robótico. Muchos controladores[107] industriales tienen ya controladores que cumplen con esta especificación, lo que en principio los hace compatibles para poder trabajar a través de ROS.
- Inspección final: el último conjunto de trayectorias definidas se realiza sobre el perfilómetro láser, donde los distintos perfiles capturados se registran en una única nube de puntos sobre la que es posible realizar comprobaciones de la calidad final tras haber realizado el proceso de acabado.

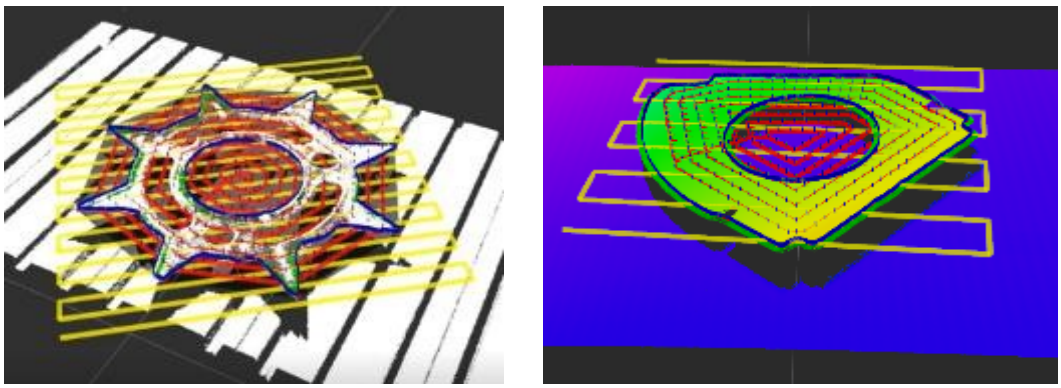


Figura 8. Los distintos colores muestran las distintas trayectorias generadas de forma automática para distintas piezas.

5.3. Celda robotizada de Inspección Óptica

PILZ ha llevado a cabo el desarrollo de una celda autoportante para la inspección óptica en 2D de dispositivos electrónicos. El principal objetivo de este proyecto ha sido desarrollar una máquina para la inspección y control de calidad de componentes en fábrica aprovechando las capacidades de ROS para el desarrollo de su software de control. En paralelo, se ha puesto especial atención en el cumplimiento del marco legal aplicable (Directiva 2006/42/CE o directiva de máquinas, entre otras) cumpliendo los requisitos de seguridad derivados de los estándares harmonizados en la directiva (ISO 10218-2, ISO 13849-1, ISO13855, IEC60204, entre otros). Con ello se ha dotado a la máquina del marcado CE necesario para su comercialización y puesta en producción.

Los elementos principales que configuran esta máquina son (ver Figura 9): una cámara de visión COGNEX In-Sight 7800 y los siguientes componentes fabricados por PILZ, el brazo PRBT-6, un PLC de seguridad y proceso PSS4000, y diferentes elementos de seguridad (barreras ópticas, *interlocks*, selector de modos de operación, entre otros).

A nivel práctico, esta celda ha permitido pasar de un sistema completamente estático y limitado a la hora de adaptarse a nuevas referencias, a una solución mucho más flexible y modular cuya funcionalidad es fácilmente expandible gracias al uso de ROS.

El sistema de control de esta máquina está basado íntegramente en ROS (Figura 10), con el paquete *pilz_robots* para el control del manipulador PRBT[108] y el paquete de generación de trayectorias determinísticas estándar de robot industrial (PtP, Lin, Circ) *pilz_industrial_motion*[109] (actualmente incorporado en MoveIt!), integrados en el sistema junto con otros paquetes disponibles a nivel de comunidad, como por ejemplo *ros_opcua*[110].

---

[106] https://wiki.ros.org/Industrial/Industrial_Robot_Driver_Spec
[107] http://wiki.ros.org/Industrial/supported_hardware
[108] http://wiki.ros.org/pilz_robots
[109] http://wiki.ros.org/pilz_industrial_motion
[110] http://wiki.ros.org/ros_opcua_communication

Además, se han desarrollado nuevos paquetes para aquellas funcionalidades para las que no había paquetes ROS disponibles, entre los que se encuentran, por ejemplo, un paquete para el control de la máquina a alto nivel mediante un enfoque de máquina de estados, dotado de características específicas en el control de maquinara industrial: gestión de eventos relativos a la seguridad (cruce de barreras, paros de emergencia, etc.), gestión de alarmas, mensajes al operador, además de la gestión del proceso, y paquetes para la interacción con otros dispositivos, como un lector de código de barras, interfaz con el PLC y un paquete encargado de la parametrización y control del sistema de visión Cognex. Todo ello se complementa con una interfaz de visualización y control (HMI) basada en el software PASVisu[111] de PILZ, que hace uso de OPC-UA para el intercambio de datos con el PLC de seguridad y el sistema ROS.

a) b)

Figura 9. a) Máquina en producción, y, b) detalle del robot con la cámara embarcada.

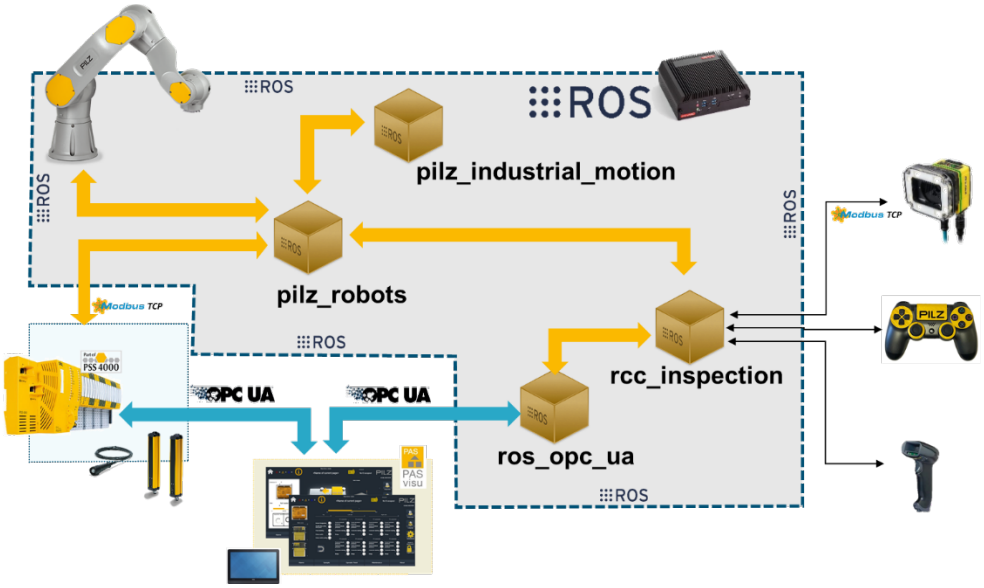


Figura 10. Arquitectura del software basado en ROS.

[111] https://www.pilz.com/es-ES/products/software/visualisation-software/pasvisu-hmi-software

En cuanto a prestaciones, esta célula es capaz de soportar hasta 43 referencias distintas con hasta 37 puntos de inspección por referencia, hecho que demuestra el nivel de flexibilidad adquirido, en parte debido a un paquete dedicado a la preparación de nuevas referencias mediante la edición de ficheros de configuración YAML soportada en una aplicación GUI y un paquete para la definición de las posiciones del robot.

Actualmente, esta máquina se encuentra en la planta de producción de PILZ en Östfildern, Alemania. Si bien, tal como se mencionó anteriormente, la implantación de ROS en la industria está a día de hoy en fase de desarrollo, la implementación de esta máquina demuestra que la implantación de ROS en procesos productivos es completamente viable y segura. Por otro lado, el uso de ROS como plataforma para el desarrollo del software de control de la máquina provee una gran flexibilidad a la hora de programar y ampliar el sistema, a la vez que permite agilizar el proceso de integración gracias a su arquitectura modular basada en paquetes, en su mayoría *open-source*.

5.4. Sharework: Colaboración eficaz entre humanos y robots en la industria

En el escenario actual de crecimiento de la robótica colaborativa, el proyecto europeo Sharework[112], liderado por el centro tecnológico Eurecat, tiene como finalidad la implementación de tecnologías innovadoras de inteligencia artificial para posibilitar la colaboración efectiva de robots con trabajadores, priorizando la seguridad y la ergonomía de los operarios. En concreto, en el proyecto se ha desarrollado un software flexible formado por 14 módulos de software para proporcionar a los robots la inteligencia necesaria para trabajar en cooperación con los operarios sin necesidad de barreras físicas de protección.

El sistema Sharework ha sido diseñado desde origen como un sistema nativo de ROS y los módulos se relacionan entre sí utilizando las interfaces propias de ROS: *topics*, servicios y acciones. Todo el hardware implicado (por ejemplo, brazos robóticos, herramientas robóticas y ejes lineales) ha sido adaptado a ROS. Por otro lado, algoritmos basados en Inteligencia Artificial que, a priori, no estaban destinados a su uso en ROS ni en robótica, han sido adaptados y usados como base para algunos desarrollos del proyecto. El sistema modular desarrollado es capaz de comprender el entorno y las acciones humanas, gestionar dicho conocimiento con el soporte de una base de conocimiento, hacer predicciones del estado futuro del proceso productivo y sus agentes, y hacer que el robot actúe en consecuencia con el objetivo de impulsar el trabajo colaborativo entre operarios y robots y, por tanto, aumentar la productividad del proceso. En el marco del proyecto se ha probado, con éxito, el uso de múltiples cámaras y sensores, el procesamiento inteligente de datos y la incorporación de realidad aumentada y tecnología de reconocimiento de gestos y del habla, con el fin de proveer de inteligencia los robots y adaptar su labor a las necesidades de los trabajadores[113]. La Figura 11 muestra la celda colaborativa desarrollada por Eurecat para realizar remachado y encolado colaborativo para el sector ferroviario. El demostrador hace uso del sistema de visión para la monitorización de la postura y actividades del operario y la planificación adaptativa de la tarea del robot.

[112] https://sharework-project.eu/

[113] Dionisis Andronas, Angelos Argyrou, Konstantinos Fourtakas, Panagiotis Paraskevopoulos, & Sotiris Makris. (2021). Design of Human Robot Collaboration workstations – Two automotive case studies. 5th International Conference on System-Integrated Intelligence (SYSINT). https://doi.org/10.1016/j.promfg.2020.11.047

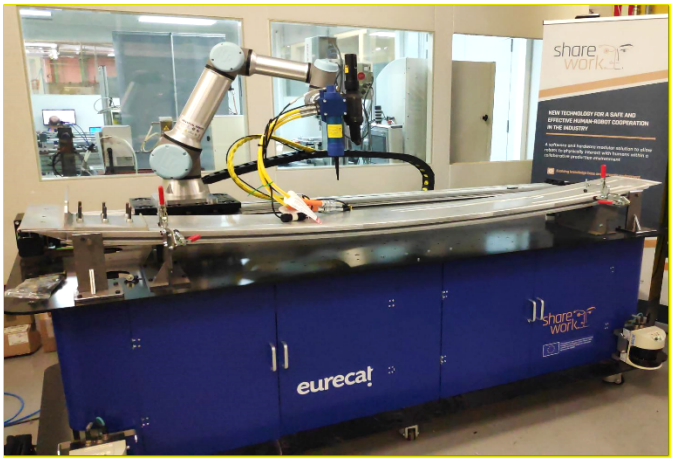


Figura 11. Celda colaborativa para el sector ferroviario usando el sistema Sharework.

El desarrollo del sistema también contempla el continuo estudio de los factores humanos, con el objetivo de adaptar y mejorar la percepción por parte del usuario y aumentar el nivel de aceptación de la nueva solución robótica por parte de los operarios. De esta forma, las industrias podrán contar con un sistema para la implementación de robótica colaborativa que permita hacer más seguros y eficientes los procesos de montaje industriales, centrándose en el trabajador y proveyéndole de un sistema que le ayude en las tareas del día a día.

A modo de ejemplo, la Figura 12 muestra un robot industrial y un operario colaborando en el montaje de las puertas de un coche, de forma segura sin necesitar barreras físicas. El robot soporta el peso de la puerta (tarea ardua y poco ergonómica), mientras el operario guía el robot en el ajuste fino de las bisagras mediante gestos manuales. El operario está en todo momento monitoreado por el sistema y recibe indicaciones de forma gráfica (a través de una Tablet) sobre las tareas a realizar y los futuros movimientos del robot. La Figura 12, además de la escena general, muestra la vista que tiene el sistema Sharework (*Safety Eye*), en la que se sobreimprimen las distintas regiones dinámicas de seguridad (gris: sin restricción, verde: velocidad y fuerza limitadas, rojo: parada de emergencia). El operario lleva unas gafas de realidad aumentada (*Operator Point Of View*) en las que recibe indicaciones visuales por parte del sistema Sharework para guiar la tarea (ver flecha verde) y, además, posibles mensajes de error y/o de seguridad.

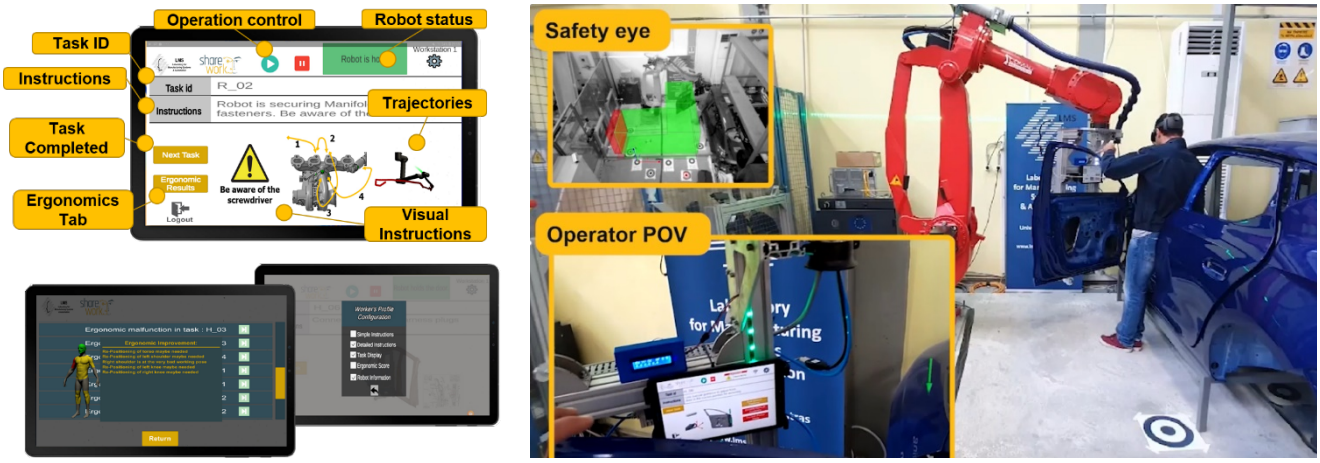


Figura 12. Montaje colaborativo de las puertas de un coche usando el sistema Sharework.

5.5. Robótica colaborativa para entornos industriales

Este caso de estudio muestra el uso de ROS como ecosistema de implementación de un proyecto que tiene como objetivo principal habilitar la realización de tareas colaborativas entre una persona y un robot manipulador de forma segura en un entorno industrial. Este proyecto, realizado por el grupo de robótica de LEITAT como parte de la agrupación Looming Factory[114], tiene como finalidad la aceleración y consolidación de tecnologías emergentes en el campo de la industria 4.0. Uno de los retos más importantes de este proyecto es plantear una estrategia de trabajo colaborativo fluido, asegurando la integridad del operario mientras comparte el espacio de trabajo con el robot. Para abordar este reto, se ha utilizado ROS en las distintas etapas de especificación e implementación del sistema. Se ha desarrollado un entorno simulado para especificar cómo debe ser el sistema que gestione la interacción operario-robot en distintas tareas productivas. Para ello, se ha utilizado el paquete *gazebo_ros_pkgs* como interfaz para el uso del simulador de Gazebo, junto con el software de MoveIt!, donde se han descrito diferentes condiciones de trabajo con distintos tipos de robots y tareas colaborativas. Por otro lado, se ha utilizado el paquete *perception_pkg* para describir al operario mediante una nube de puntos que representa la superficie 3D de su cuerpo. Finalmente, se han especificado los distintos módulos y datos del sistema en forma de nodos, mensajes, servicios y acciones de ROS, con el objetivo de:

1. Detectar la intrusión de objetos en el espacio de trabajo del robot en tiempo real.
2. Identificar la posición del operario respecto al robot.
3. Predecir las posibles interacciones físicas entre el operario-robot.
4. Determinar la respuesta del robot frente a las distintas interacciones detectadas.

Para detectar la intrusión de objetos, el sistema utiliza una cámara RGBD para obtener una nube de puntos del entorno de trabajo del robot en tiempo real. Un nodo de ROS se encarga de conectarse a la cámara y de procesar las imágenes para publicar dicha nube de puntos como mensaje de ROS. Gracias al paquete ROS-PCL, esta nube de puntos se procesa en una secuencia de nodos con distintos filtros de PCL con la finalidad de identificar aquellas superficies del entorno que son dinámicas. Dado que el robot es un objeto dinámico en el entorno, se filtran aquellos puntos generados por la superficie del robot gracias a conocimiento de la posición del robot respecto a la cámara y de la información geométrica del mismo, codificada en URDF, que es utilizada por MoveIt! para ayudar a filtrar dichos puntos. Cabe destacar que la posición del robot respecto a la cámara se obtiene mediante otro proceso de ROS que utiliza un paquete llamado *aruco_ros* que permite obtener la posición y orientación de un marcador respecto a la cámara, y gracias al conocimiento de la posición del marcador respecto a la base del robot, se tiene la transformada robot-cámara.

Para la identificación de la posición del operario se hace uso de una interfaz con ROS de un detector de personas y sus posturas basado en aprendizaje profundo llamado DensePose[115], desarrollado por FaceBook, que utiliza redes de convolución para identificar aquellas zonas de la imagen RGB que pertenecen a una parte del cuerpo. Dichas zonas se contrastan con la información de profundidad de la cámara RGBD para obtener los puntos clave en coordenadas 3D del cuerpo, que se han reducido a diversos puntos de control del tronco superior del operador. Estos puntos clave son publicados en un mensaje de ROS y son utilizados para etiquetar la nube de puntos del apartado anterior y discernir entre puntos pertenecientes al operario y puntos pertenecientes a objetos.

Para predecir la interacción física entre el operario y el robot, uno de los nodos se subscribe a la información del movimiento del robot y al mensaje de los puntos de control del operario. Con esta información se realizan diferentes cálculos de distancias y velocidades de aproximación del robot a los puntos de control del operario para determinar posibles colisiones entre ambos. La información de posibles colisiones se publica en un mensaje ROS como predicción de interacciones operario-robot.

---

[114] https://loomingfactory.upc.edu/looming-factory

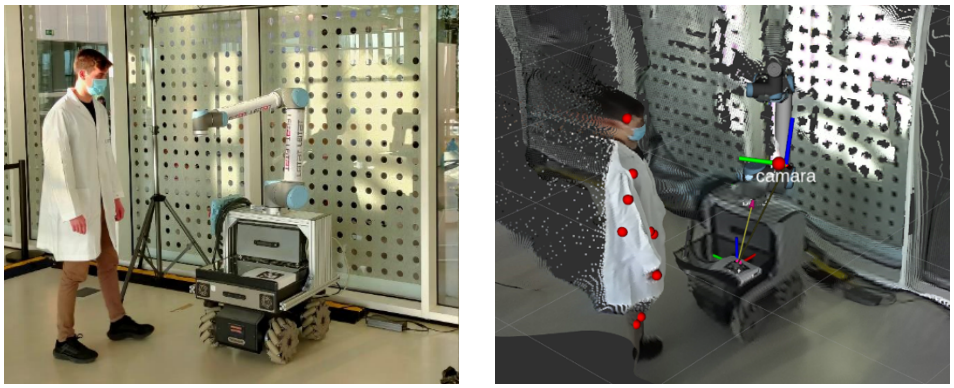
[115] Riza Alp Guler, Natalia Neverova, Iasonas Kokkinos (2018). "DensePose: Dense Human Pose Estimation In The Wild". 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Finalmente, otro nodo se subscribe al mensaje de predicción de interacción para enviar las señales de control al robot adecuadas a cada situación. Primero se define una distancia de seguridad de interacción, donde por debajo de la misma se procede a la interrupción y parada de la tarea del robot. Fuera de esta región, el comportamiento se traduce a:

1. Una parada del robot siguiendo la ISO 15066:2016, donde se calcula la distancia necesaria para realizar una parada de seguridad en función de la velocidad actual del robot, su capacidad de frenar y el tiempo de reacción del sistema para proceder a la parada del robot.
2. Una variación de la velocidad del robot en fusión de la velocidad de aproximación al punto de interacción.

La gestión del movimiento del robot se realiza gracias al uso del paquete *ros_control* que sirve de interfaz para poder gestionar el movimiento de un robot manipulador, pudiendo modificar el robot de manera sencilla en todo momento.

En este proyecto se ha desarrollado un demostrador con diferentes robots colaborativos, como el UR10 (Figura 13) y el Doosan M1013, que simula situaciones reales para hacer pruebas de validación. El demostrador se ha construido en el laboratorio de robótica de Leitat en DFactory Barcelona, como ejemplo de sistema inteligente para la colaboración persona-robot en la industria 4.0.



a) b)

Figura 13. a) Operario en área de trabajo colaborativo, b) Monitorización del operario para la predicción y evasión de colisiones

6. Conclusión

En este artículo se ha presentado ROS, acrónimo de *Robot Operating System* (Sistema Operativo para Robots), incluyendo fundamentalmente una breve historia de su origen, las principales características técnicas, nuevas extensiones como ROS2 y ROS industrial, dónde obtener información relacionada para formarse en el tema, y, finamente, varios casos de aplicación que muestras su potencial en distintos contextos.

La difusión de ROS en el entorno académico es notable y marcadamente creciente en el entono industrial, lo que deja entrever que en un futuro cercano puede ser un elemento común e indispensable en muchas aplicaciones de la robótica, cada vez más variadas y cubriendo nuevos sectores productivos y de servicios. Evidentemente, la velocidad con que se producen cambios tecnológicos puede provocar la aparición de nuevos estándares en el campo de la robótica, pero, dada su difusión actual, es previsible que ROS marque el camino a seguir, ya sea con sus actuales características o con posibles variantes derivadas de ellas.