Trabajo de Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Herramienta software de un proyector láser para aplicaciones de Realidad Aumentada en entornos industriales

Autor: Rafael Luque Berraquero
Tutor: Jesús Capitán Fernández

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Herramienta software de un proyector láser para aplicaciones de Realidad Aumentada en entornos industriales

Autor:
Rafael Luque Berraquero

Tutor:
Jesús Capitán Fernández
Profesor Contratado Doctor

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo de Fin de Máster: Herramienta software de un proyector láser para aplicaciones de Realidad Aumentada en entornos industriales

Autor: Rafael Luque Berraquero

Tutor: Jesús Capitán Fernández

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia.

A mis tutores.

A mis compañeros.

Agradecimientos

Primero de todo, quiero agradecer a mi familia por su apoyo incondicional a lo largo de los años y su ánimo en los buenos momentos, pero sobre todo en los difíciles.

Dar las gracias a Eduardo Ferrera, Jefe de la Unidad de Automatización y Robótica en el Centro Avanzado de Tecnologías Aeroespaciales (FADA-CATEC), y tutor externo de este trabajo, por darme la oportunidad de llevar a cabo este proyecto dentro del centro.

Dar las gracias, también, a mi tutor académico, Jesús Capitán, por su disposición a ayudarme y aconsejarme en todo lo que he necesitado. Y al resto de mis profesores porque de una manera u otra todos tienen una parte en este documento.

A mis compañeros de Unidad y a todos mis compañeros de CATEC en general, por las buenas amistades que estoy formando allí, y, sobre todo, a mi compañera Inés Lara, por guiarme y transmitirme su extraordinario conocimiento en el campo de la robótica y el desarrollo de software basado en ROS.

Finalmente, agradecer al consorcio del proyecto *"ROSIN - ROS-Industrial Quality-Assured Robot Software Components",* por financiar parte de este proyecto (Más información: http://rosin-project.eu/).

Rafael Luque Berraquero

Sevilla, 2020

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732287.

Resumen

Este trabajo se ha desarrollado en FADA-CATEC y forma parte del conjunto de 'Proyectos Técnicos Enfocados (FTP)' admitidos en la tercera convocatoria de financiación en cascada abierta en el marco del proyecto europeo: *'ROSIN - ROS-Industrial Quality-Assured Robot Software Components'*.

El objetivo de este proyecto consiste en desarrollar un paquete software basado en ROS que permita la comunicación, control e integración de un proyector láser, concretamente el modelo ZLP1 de la marca Z-LASER, convirtiéndolo en un dispositivo totalmente compatible con este framework. El paquete permitirá al usuario de ROS operar este dispositivo y facilitará el desarrollo de aplicaciones de realidad aumentada más avanzadas que proporcionen apoyo a los operarios en sus tareas dentro de los procesos de fabricación industriales, impulsando así el crecimiento de la comunidad de ROS-Industrial en este campo.

Para ello, se desarrollan diferentes herramientas, en forma de subpaquetes independientes, que conforman el paquete general completo, a saber: un primer subpaquete que recoge la declaración de mensajes y servicios comunes, otro para la comunicación y control del proyector, otro para la interfaz de usuario, y, por último, otro para el visualizador de proyecciones. Al mismo tiempo, se aplican también las diferentes prácticas y procedimientos para cumplir con los requisitos de garantía de calidad del software que exige el proyecto ROSIN a los FTP financiados, como pueden ser: los tests automáticos, la integración continua, la cobertura de código, la documentación del paquete, entre otros.

Finalmente, se exponen una serie de conclusiones y posibles trabajos o aplicaciones futuras que se pueden desarrollar a partir del paquete obtenido en este trabajo.

Abstract

This study has been developed at FADA-CATEC, being one of the "Focused Technical Projects (FTP)" accepted in the third open call for cascade funding in the framework of the European project: "*ROSIN - ROS-Industrial Quality-Assured Robot Software Components*."

The objective of this project is to develop a ROS software package that enables the communication, control and integration of a laser projector -specifically the ZLP1 model of the Z-LASER brand- turning it into a fully compatible device within this framework. The package will allow the ROS user to operate the above-mentioned gadget and ease the development of more advanced augmented reality applications. As a result, it will greatly provide support to operators in their tasks within the industrial manufacturing processes, thus boosting the ROS-Industrial community growth around this same field of study.

To this end, different tools have been accordingly developed in the form of independent sub-packages which create the complete general package: a first sub-package that includes the declaration of common messages and services, a second for the communication and control of the projector, a third for the user interface, and finally another for the projection visualizer. At the same time, different practices and processes are also applied to achieve the software quality assurance required by the ROSIN project for funded FTPs -automatic tests, continuous integration, code coverage, and package documentation among others.

To conclude, a series of conclusions and possible future projects or applications that can be developed from the package are as well presented for further investigation.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xiv**

Índice de Tablas **xvi**

Índice de Figuras **xviii**

1 Introducción **1**

2 Marco Contextual **4**

3 Estado Del Arte **11**
3.1 Tecnologías *12*
3.2 Aplicaciones *15*
3.3 Proyección láser aplicada a la fabricación *16*

4 Herramientas y Equipos **19**
4.1 Modelo ZLP1 *19*
4.2 Apache Thrift *21*

5 Desarrollo **25**
5.1 z_laser_msgs *26*
5.2 z_laser_zlp1 *28*
5.3 z_laser_viz *30*
5.3.1 TF frames 31
5.3.2 Visualization Markers 31
5.4 z_laser_gui *34*

6 Garantía de Calidad **38**
6.1 Control de versiones *39*
6.1.1 Git Branching Workflow 41
6.2 Guías de estilo *41*
6.2.1 Escáner de código 43
6.3 Automatización de tests *43*
6.3.1 Tests unitarios (nivel 1 y nivel 2) 45
6.3.2 Tests de integración (nivel 3) 46
6.4 Integración Continua *46*
6.4.1 Ejecución automática de los tests 48
6.5 Cobertura de código *49*
6.6 Licencia *50*
6.7 Documentación *51*
6.7.1 ROS Wiki del paquete 51
6.7.2 Code API 53

6.8 Lanzamiento *54*

7 Conclusiones **56**

Referencias **58**

Glosario **60**

ÍNDICE DE TABLAS

Tabla 3–1. Ventajas y desventajas de las tecnologías de RA. 14

Tabla 4–1. Especificaciones técnicas del proyector láser ZLP1. 19

Tabla 4–2. Área de trabajo en función de la distancia de instalación, con un ángulo de apertura máximo de 60º. 21

ÍNDICE DE FIGURAS

Figura 1-1. Logo de la marca Z-LASER. 1

Figura 1-2. Propuesta de logo para el proyecto. 3

Figura 2-1. Logo de ROS. 4

Figura 2-2. Logo de ROS-Industrial. 5

Figura 2-3. Logo de del proyecto ROSIN. 6

Figura 2-4. Ejemplos de Convocatorias FSTP derivadas de proyectos europeos en 2019. Se localizan en el mapa situadas por países según la ubicación del coordinador del consorcio del proyecto: CloudiFacturing (Alemania, Fraunhofer-Gesellschaft), MIDIH (Bélgica, EIT Digital), COVR (Dinamarca, Teknologisk Institut), inDemand (España, TICBioMed), L4MS (Finlandia, Teknologian tutkimuskeskus VTT Oy), RIMA (Francia, Commissariat à l'énergie atomique et aux énergies alternatives), ESMERA (Grecia, Panepistimio Patron), TERRINet (Italia, Scuola Superiore Sant'Anna), Cross4Health (Noruega, Norway Health Tech), ROSIN (Países Bajos, Technische Universiteit Delft), LEDGER (Polonia, Fundingbox Accelerator sp. z o.o) y SCIFI (Reino Unido, Centre for Economic Policy Research). 7

Figura 2-5. Logos de FADA-CATEC. 8

Figura 2-6. Célula robótica en las instalaciones de CATEC. 10

Figura 3-1. Continuo de Virtualidad. 12

Figura 3-2. Aplicación de realidad aumentada basada en proyección tradicional para ayudar al operario en el proceso de ensamblaje. 12

Figura 3-3. Aplicación de realidad aumentada basada en proyección láser para ayudar al operario en el proceso de mecanizado. 13

Figura 3-4. Aplicación de realidad aumentada basada en VST. 13

Figura 3-5. Aplicación de realidad aumentada basada en OST. 14

Figura 3-6. Ejemplos de uso de proyección láser en procesos industriales: a) en el proceso de mecanizado de aeronaves, b) proceso de montaje de soportes en la fabricación de vehículos, c) perfil de corte de pieza de madera, d) posicionamiento de pedidos sobre el carro de transporte. 18

Figura 4-1. Modelo proyector láser ZLP1 de la marca Z-LASER. 19

Figura 4-2. Representación de las dimensiones características en un sistema de proyección láser. 21

Figura 4-3. Flujo de operación para la programación de una interfaz basada en Apache Thrift. 22

Figura 4-4. Arquitectura de comunicación de Apache Thrift. 23

Figura 4-5. Arquitectura de comunicación del sistema de proyección Z-LASER. 24

Figura 5-1. Arquitectura del paquete Z_LASER_PROJECTOR. 25

Figura 5-2. Proyección de un sistema de referencia (a la izquierda) y los diferentes tipos de elementos de proyección (a la derecha). 30

Figura 5-3. Representación de los sistemas de referencia (‘frames’) definidos en el visualizador y publicados mediante el paquete de ROS-tf. 31

Figura 5-4. Proyección de los sistemas de referencia y los diferentes tipos de elementos de proyección en el

visualizador. 33

Figura 5-5. Ejemplo de uso del visualizador en el que se proyecta el contorno del hueco en una pieza aeronáutica. 34

Figura 5-6. Interacción con la ROS API del paquete desde el terminal. 34

Figura 5-7. Vista de las opciones de objetos de la herramienta QtDesigner. 35

Figura 5-8. Menú general de la GUI. 36

Figura 5-9. Submenús dedicados a los sistemas de coordenadas. 36

Figura 5-10. Submenús dedicados a los elementos de proyección. 37

Figura 5-11. Ventana emergente con un mensaje de error producido al iniciar la proyección sin haber definido ninguna figura. 37

Figura 6-1. Propiedades de calidad del software por la ISO/IEC 25010:2011[24]. 38

Figura 6-2. Ejemplo de muestra de los resultados de cobertura de código en la plataforma Codecov. A la izquierda se muestran los resultados en un gráfico estilo 'sunburst', y a la derecha se muestran en forma de valores porcentuales para cada archivo. 50

Figura 6-3. Página Wiki del paquete Z_LASER_PROJECTOR. 52

Figura 6-4. Ejemplo de CODE API del código del paquete Z_LASER_PROJECTOR utilizando la herramienta Sphinx y el estilo Google Docstring. 54

Figura 6-5. Ejemplo de página Wiki enlazada al repositorio del paquete una vez se ha añadido al ROS Index. 55

Figura 7-1. Ejemplo de figura en archivo gráfico con formato DWG. 56

Figura 7-2. Aplicación de realidad aumentada basada en proyección láser. Proyecto AEROPAINT (FADA-CATEC). 57

1 INTRODUCCIÓN

Este trabajo tiene como objetivo crear un paquete software, basado en el framework de ROS[1], que englobe las distintas funcionalidades de un proyector láser industrial, así como la comunicación y control del dispositivo, de manera que sea completamente compatible e integrable con el resto de herramientas del entorno de ROS y ROS-Industrial.

El proyector láser que se va a utilizar es el modelo ZLP1 de la marca Z-LASER[2]. Se trata de un dispositivo diseñado para proporcionar apoyo a los operarios, de manera fiable y segura para los ojos, en las tareas de procesos industriales de producción y fabricación, a través de soluciones de realidad aumentada.

Figura 1-1. Logo de la marca Z-LASER.

Esta tecnología de realidad aumentada es una herramienta ampliamente utilizada, totalmente relacionada con la Industria 4.0. Está presente en múltiples sectores industriales como el aeronáutico, naval, automotriz, construcción, metalúrgico, electrónica, y en otros no industriales como la medicina, la educación o el marketing. Entre las aplicaciones industriales, esta tecnología adquiere especial relevancia ya que puede ser empleada en diferentes etapas de los procesos de producción: i) visualización del flujo de trabajo como guía óptica para los operarios en el ensamblaje de piezas; ii) señalización de las dimensiones y tolerancias de las piezas en los procesos de mecanizado; iii) indicación en logística, para aumentar la productividad; entre otras.

Esta aplicabilidad se traduce en una previsión creciente de la realidad aumentada en el mercado, concretamente, MarketsAndMarkets[3], prevé un crecimiento de 4.21 mil millones de dólares en 2017 a 60.55 mil millones en 2023 en aplicaciones de realidad aumentada. Sin embargo, a pesar de esto, el principal problema que plantea la Industria 4.0 a la hora de implementar soluciones basadas en realidad aumentada está relacionado con los costes en implementación de la tecnología. Muchas aplicaciones que se llevan a cabo mediante gafas de realidad aumentada o mediante tablets requieren grandes esfuerzos económicos, lo que hace que la inversión correspondiente sea casi inaccesible para Pequeñas y Medianas Empresas (PYMEs).

Este no es el caso de los sistemas de proyección láser, en los que muchas aplicaciones sencillas pero potentes, pueden mejorar ampliamente los procesos de producción sin necesidad de grandes inversiones. De hecho, MarketDataForecast[4], espera que el mercado global de proyección láser también se desarrolle de manera notable, creciendo desde 5.37 mil millones en 2020, hasta alcanzar un valor de 12.57 mil millones de dólares para 2026, lo que supone una tasa de crecimiento anual compuesto (CAGR) del 17.29% para el período previsto.

El desarrollo de un paquete lo suficientemente flexible para proporcionar el control a alto nivel de un sistema de proyección láser de carácter industrial que resuelva la compatibilidad y que permita la integración del dispositivo en un entorno tan extendido y versátil como es el de ROS, fomentará la creación de nuevas aplicaciones de realidad aumentada que hagan uso de este dispositivo, impulsadas por el ahorro que supone en gran parte del trabajo de implementación. Además, ayudará al crecimiento de la comunidad industrial de ROS

[1] https://www.ros.org/
[2] https://z-laser.com/en/
[3] https://www.marketsandmarkets.com/Market-Reports/augmented-reality-virtual-reality-market-1185.html
[4] https://www.marketdataforecast.com/market-reports/laser-projection-market

en otros sectores, alentando a otros fabricantes a realizar procesos de “rossificación” similares.

Al mismo tiempo, se espera que la combinación de esta tecnología junto con los sensores que ya funcionan en ROS mejore aún más las capacidades de la Industria 4.0. Como ejemplo simple, un lidar[5] en combinación con las capacidades de ROS-tf podría hacer uso del proyector para rastrear piezas a lo largo de una habitación, mostrando en su superficie sus destinos en un almacén.

De forma general, las tareas a desempeñar para alcanzar los objetivos de este trabajo son:

- Revisar e investigar desarrollos basados en proyección láser, relacionados y no relacionados con la comunidad de ROS.
- Estudiar la documentación e información proporcionada por el fabricante y diseñar la arquitectura software del paquete.
- Desarrollar una biblioteca ROS flexible, capaz de comunicarse con el proyector bajo sus estándares y protocolos.
- Desarrollar una biblioteca ROS que proporcione funciones y servicios para controlar el proyector, y validarla mediante pruebas en el laboratorio.
- Desarrollar las funcionalidades para la definición, creación, proyección y gestión de sistemas de coordenadas y elementos de proyección admitidos por el dispositivo, así como la calibración automática mediante el escaneo de reflectores.
- Construir un visualizador de los elementos de proyección definidos en el que se pueden importar modelos 3D de piezas y obtener una previsualización del resultado final.
- Diseñar una interfaz gráfica (GUI) para operar el proyector, evitando que el usuario necesite interactuar directamente con el terminal.
- Aplicar las mejores prácticas de desarrollo de software industrial de calidad a estas bibliotecas.
- Probar la solución desarrollada en un entorno realista fuera de las condiciones controladas del laboratorio, para demostrar la aplicabilidad de la tecnología a la industria en un caso de uso relacionado con la industria aeroespacial, incluyendo las posibles mejoras encontradas durante esta fase de validación.

Con estos objetivos, los posibles desafíos a los que se enfrenta el proyecto son:

i) se debe lograr una "rossificación" del dispositivo a fin de incorporar sus funcionalidades en el entorno de ROS;

ii) la biblioteca que se desarrolle debe respetar los mensajes estandarizados de ROS, como son los *visualization_msgs/Marker*;

iii) se debe realizar un estudio de los límites de la información proyectable (la visualización de grandes cantidades de información hace que los proyectores láser parpadeen);

iv) se debe perseguir la mayor usabilidad, a fin de aumentar el TRL del paquete resultante.

En conclusión, este proyecto propone el desarrollo de una biblioteca para controlar el proyector ZLP1 desde un alto nivel, exponiendo todas sus funcionalidades como topics y servicios del marco de ROS.

[5] Acrónimo del inglés LIDAR (Light Detection and Ranging o Laser Imaging Detection and Ranging) es un dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado.

Figura 1-2. Propuesta de logo para el proyecto.

2 MARCO CONTEXTUAL

ROS (del inglés 'Robot Operating System') es un framework para el desarrollo de software para robots que provee las funcionalidades de un sistema operativo en un clúster heterogéneo. ROS fue desarrollado originalmente en 2007 por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto de un robot con inteligencia artificial (STAIR). Desde 2008, el desarrollo continuó primordialmente en Willow Garage, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado. A partir de 2013, tras la absorción de Willow Garage por parte de sus fundadores, hasta día de hoy, el desarrollo ha estado a cargo de la 'Open Source Robotics Foundation' (OSRF).

Figura 2-1. Logo de ROS.

Este framework provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidades de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. La librería está orientada para un sistema UNIX (Ubuntu de Linux) aunque también se está adaptando a otros sistemas operativos como Fedora, Mac OS X, Arch, Gentoo, OpenSUSE, Slackware, Debian o Microsoft Windows, considerados como "experimentales".

Se compone de dos partes básicas: la parte del sistema operativo, ROS, como se ha descrito anteriormente, y ros-pkg, una suite de paquetes aportados por la contribución de usuarios (organizados en conjuntos llamados pilas o 'stacks') que implementan funcionalidades tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc.

ROS es un software desarrollado bajo términos de licencia BSD (Berkeley Software Distribution), lo que quiere decir que es libre (del inglés 'open source') y, por tanto, accesible al público: todos pueden ver, modificar y distribuir el código de la forma que consideren conveniente, ya sea para uso comercial o investigador. El objetivo principal es permitir a los desarrolladores de software construir aplicaciones robóticas más capaces, de forma rápida y fácil en una plataforma común.

Este tipo de software 'open source' se desarrolla de manera descentralizada y colaborativa, así que depende en gran medida de la revisión entre usuarios y la producción de la comunidad. Por otro lado, suele ser más económico, flexible y duradero que sus alternativas propietarias, ya que las encargadas de su desarrollo son las comunidades y no un solo autor o una sola empresa. De hecho, ROS, es posiblemente el conjunto de software de código abierto más adoptado por los fabricantes de robots de hoy en día. Se puede encontrar comúnmente en robots industriales, robots móviles autónomos terrestres (UGV, 'Unmanned Ground Vehicle') y aéreos (UAV, 'Unmanned Aerial Vehicle'), robots de servicios, etc.

Sin embargo, como se ha comentado, la principal característica del software libre es que el desarrollo y la disponibilidad de herramientas y soluciones depende en gran medida de los aportes realizados por la comunidad de usuarios. Bajo esta premisa, en 2012, surge ROS-Industrial[6], un proyecto de código abierto cuyo objetivo es extender y ampliar el horizonte de aplicación de las capacidades avanzadas de ROS a la

[6] https://rosindustrial.org/

automatización y robótica dentro del ámbito industrial.

El proyecto comenzó como un esfuerzo colaborativo entre Yaskawa Motoman Robotics, Southwest Research Institute y Willow Garage, y ha crecido con éxito hasta convertirse en una iniciativa mundial respaldada por tres consorcios regionales: el consorcio ROS-Industrial 'Americas' (liderado por SwRI), el consorcio ROS-Industrial 'Europe' (liderado por Fraunhofer IPA) y el consorcio ROS-Industrial 'Asia-Pacific' (liderado por el Advanced Remanufacturing and Technology Centre, ARTC, y la Universidad Tecnológica de Nanyang, UTN). ROS-Indusrial cuenta, además, con el apoyo de un consorcio internacional de miembros de la industria y la investigación. Estos consorcios apoyan a la comunidad global ROS-Industrial a través de formación, proporcionando apoyo técnico, garantizando el cumplimiento de las normas de calidad de código apropiadas para un producto de software industrial, estableciendo la hoja de ruta futura para ROS-Industrial, proporcionando mecanismos para la comercialización y distribución oficial de código a un público más amplio, así como la realización de proyectos industriales para desarrollar nuevas capacidades.

Hasta ahora, los procesos de fabricación se han basado en tareas simples y repetitivas realizadas por robots, pero en los últimos años ha habido una creciente demanda de robots más dinámicos e inteligentes que puedan resolver tareas más complejas. ROS-Industrial ofrece herramientas y programas de alto nivel que pueden hacer frente a estos desafíos, utilizando una plataforma común para facilitar la transferencia de tecnología de los laboratorios de investigación a la industria.

Figura 2-2. Logo de ROS-Industrial.

Para ello, ROS-Industrial dispone de un repositorio[7] en el que se incluyen, entre otras herramientas, interfaces para manipuladores industriales comunes, pinzas, sensores, redes de dispositivos, etc. También proporciona bibliotecas de software para la calibración automática de sensores 2D/3D, procesos de trazado y planificación del movimiento, aplicaciones como Scan-N-Plan, herramientas para desarrolladores como el plugin QtCreator, un currículo de entrenamiento específico para las necesidades de los fabricantes, etc.

De esta manera la industria puede beneficiarse de la investigación y el desarrollo creado alrededor de ROS, proporcionando excelentes herramientas de visualización e interacción hombre-máquina (HMI), que pueden facilitar el uso de estos sistemas robóticos por personal no experto. A su vez, esto favorece la estandarización, que se traduce en tiempos de despliegue más rápido de los sistemas robóticos avanzados en los lugares de fabricación. Por su parte, ROS también se beneficia de la simbiosis creada gracias a ROS-Industrial, por ejemplo, en forma de nuevos controladores creados por fabricantes de brazos robóticos, así como un número creciente de herramientas que están siendo mejoradas por ROS-Industrial, como herramientas de navegación, percepción, manipulación y calibración.

Sin embargo, a pesar de que ROS y su filial ROS-Industrial 'Europe' es bien conocida, su potencial industrial está subestimado, siendo las dos principales críticas: 1) ¿está la calidad a la altura de la industria?, y 2) ¿hay suficiente interés por parte de la industria europea para justificar una inversión? La respuesta es "sí" y "sí", en parte, ya que las dos preguntas se mantienen en un punto muerto: una mayor mejora de la calidad requiere una inversión industrial, y viceversa.

Con el fin de resolver este punto muerto y reforzar el papel de la UE dentro de ROS-Industrial consolidando así el liderazgo europeo en tecnologías de fabricación, se impulsa el proyecto ROSIN[8], un proyecto financiado por el programa de investigación e innovación Horizonte 2020 de la Unión Europea en virtud del acuerdo de subvención número 732287 con un presupuesto total de alrededor de 7,5 M€.

[7] https://github.com/ros-industrial/
[8] https://cordis.europa.eu/project/id/732287

Figura 2-3. Logo de del proyecto ROSIN.

El objetivo de este proyecto es mejorar en gran medida la disponibilidad de componentes software inteligentes de alta calidad para la robótica industrial europea, haciendo que ROS-Industrial sea mejor y aún más amigable para las empresas. Para ello, los socios del proyecto están llevando a cabo tres actividades principalmente[9]:

- Garantizar la calidad del software listo para uso industrial. ROSIN introduce una innovación revolucionaria en las pruebas automatizadas de calidad del código, lideradas por la Universidad de TI de Copenhague, complementadas con una completa paleta de medidas de garantía de calidad que incluyen novedosas pruebas de integración continua 'model-in-the-loop' con robots ABB.
- Asignar parte del presupuesto del proyecto a usuarios y desarrolladores de aplicaciones relacionadas con la industria europea a través de los denominados Proyectos Técnicos Enfocados (Focused Technical Projects, FTP). La experiencia demuestra que la industria financiará los desarrollos industriales de ROS, pero sólo después de que hayan tenido éxito. Por este motivo, ROSIN proporciona prefinanciación para los desarrolladores que se recuperará en un futuro fondo rotatorio para perpetuar el mecanismo, tratando de aliviar el punto muerto mencionado anteriormente.
- Proporcionar actividades de educación de manera que ROS-Industrial alcance una masa crítica con un mayor crecimiento autopropulsado resultante en un estándar industrial de código abierto de alta calidad ampliamente adoptado.

El mecanismo utilizado para financiar el desarrollo de estas aplicaciones, en forma de FTPs, mencionado en la segunda tarea del proyecto ROSIN, es del tipo: Financiación en Cascada (Cascade Funding) o, también denominado, FSTP (Financial Support to Third Parties). Se trata de un mecanismo o instrumento de la Comisión Europea para distribuir fondos públicos con el fin de ayudar a los beneficiarios, como empresas emergentes ('start-ups'), empresas en expansión ('scale-ups'), PYMEs y/o Mid-Caps[10], en la adopción o desarrollo de innovación digital.

Este método de financiación tiene como objetivo simplificar los procedimientos administrativos, creando un sistema de solicitud sencillo para las PYMEs al permitir que algunos proyectos financiados por la UE puedan emitir, a su vez, otras convocatorias abiertas ('Open Calls'). Este plan se basa en el modelo de los estudiantes Erasmus y fue introducido por primera vez por la Comisión Europea en el Programa Marco de Investigación e Innovación (2014-2020), Horizonte 2020 (H2020).

Previamente han existido otras prácticas de este tipo dentro del Séptimo Programa Marco de Investigación e Innovación (7PM), como es la Asociación Público-Privada de Internet del Futuro (FI-PPP) y la Innovación en TIC para PYMEs de fabricación (I4MS). Sin embargo, el procedimiento para prestar apoyo financiero a terceros en el marco de Horizonte 2020 es significativamente diferente, ya que, en estas primeras experiencias, los terceros se convertían en beneficiarios del proyecto por el que fueron seleccionados y se inscribían en un contrato con la Comisión Europea. En cambio, los terceros que reciben financiación en cascada en el marco de Horizonte 2020, tienen un contrato con el consorcio del proyecto financiado por la Unión Europea, del cual se convierten en terceros. Esto significa que este consorcio es responsable, ante la Comisión Europea, de los terceros a los que presta apoyo financiero, y, por tanto, no es necesaria una validación jurídica y financiera de esos terceros, facilitando la apertura de convocatorias FSTP en el seno de proyectos europeos que cumplen el denominado Anexo K ("Acciones de apoyo financiero a terceros")[1] de las normas de participación de Horizonte 2020.

Estos 'Open Calls', de proyectos financiados por la UE, suelen ser en régimen de concurrencia competitiva y

[9] https://www.rosin-project.eu/

[10] Ver: https://ec.europa.eu/growth/smes/sme-definition_en o 'Commission Recommendation of 6 May 2003 concerning the definition of micro, small and medium-sized enterprises (2003/361/EC)'

tienen una dimensión y objetivos europeos:

- seleccionar empresas tecnológicas de nueva creación o de ampliación de escala con fines de aceleración o incubación (normalmente sin capital social),
- el apoyo a proyectos piloto, demostradores y/o experimentos sobre una tecnología o marco innovador específico (generalmente con la participación de empresas de nueva creación o PYMES), o
- integrando más participantes en el proyecto para ampliar su alcance o abordar tareas específicas.

El apoyo que ofrecen este tipo de convocatorias suele consistir en financiación (normalmente en el rango de 50.000 a 150.000 euros), pero también puede consistir en vales (‘vouchers’) para servicios de soporte o para utilizar instalaciones de pruebas.

La financiación en cascada se utiliza cada vez más en toda la prioridad "Tecnologías de la información y la comunicación" en Horizonte 2020 y contribuye "a la transferencia y explotación de conocimientos, la adopción de tecnologías y la construcción y ampliación de ecosistemas en esas zonas".

Como se ha comentado, las convocatorias de este tipo, derivadas de proyectos europeos diferentes, presentan líneas principales comunes en sus bases regulatorias, sin embargo, estas suelen exigir requisitos de aplicación diferentes, como pueden ser, por ejemplo: la cantidad máxima de la financiación, porcentajes, presupuesto y duración máxima de los proyectos financiados, tipos de beneficiarios, número de beneficiarios, beneficiarios individuales o en consorcio, entre otros.

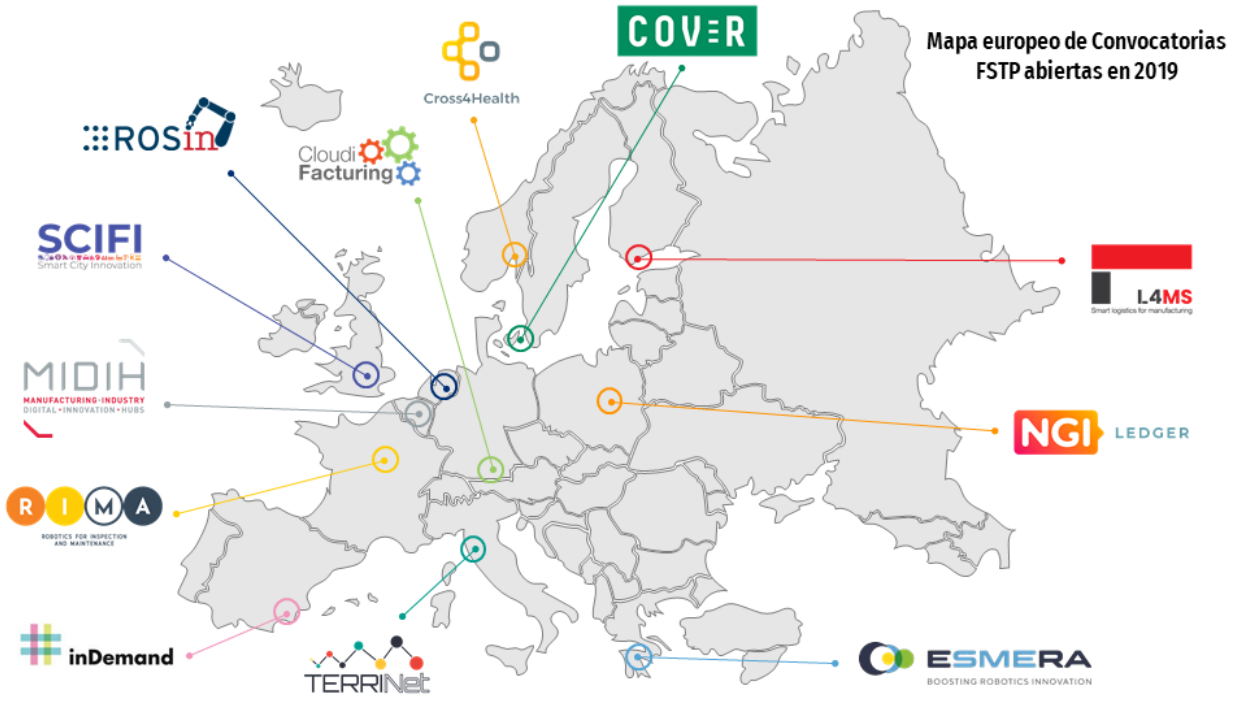


Figura 2-4. Ejemplos de Convocatorias FSTP derivadas de proyectos europeos en 2019. Se localizan en el mapa situadas por países según la ubicación del coordinador del consorcio del proyecto: CloudiFacturing (Alemania, Fraunhofer-Gesellschaft), MIDIH (Bélgica, EIT Digital), COVR (Dinamarca, Teknologisk Institut), inDemand (España, TICBioMed), L4MS (Finlandia, Teknologian tutkimuskeskus VTT Oy), RIMA (Francia, Commissariat à l'énergie atomique et aux énergies alternatives), ESMERA (Grecia, Panepistimio Patron), TERRINet (Italia, Scuola Superiore Sant'Anna), Cross4Health (Noruega, Norway Health Tech), ROSIN (Países Bajos, Technische Universiteit Delft), LEDGER (Polonia, Fundingbox Accelerator sp. z o.o) y SCIFI (Reino Unido, Centre for Economic Policy Research).

En el caso de la convocatoria “ROSIN Call for Focused Technical Projects”, los requisitos principales, los cuales se pueden encontrar en la guía de aplicación[2], son:

- Fecha límite de entrega (‘deadline’) para la tercera convocatoria: 13 de septiembre de 2019.
- Las propuestas FTP serán enviadas por una sola entidad o un consorcio de "solicitantes", uno de ellos será el líder (‘champion’).
- Beneficiarios: la entidad o entidades deben estar establecidas en uno de los Estados Miembros de la

UE o países asociados, incluyendo, pero no limitadas a:

- Universidades
- Organismos de investigación
- Start-ups
- PYMEs
- Grandes empresas

- Los participantes pueden solicitar la financiación presentando una memoria/propuesta en la que se describa el objetivo del FTP, el plan técnico para lograrlo y una estimación de los costos correspondientes.
- El apoyo financiero proporcionado por ROSIN ascenderá a un 33% del costo total de desarrollo de la FTP hasta un máximo de 100.000 euros, comprometiéndose las organizaciones involucradas a financiar la parte restante.
- Los FTP tienen una duración prevista de 12 meses con tres hitos.

Ejemplo de Organismo de Investigación es el Centro Avanzado de Tecnologías Aeroespaciales (CATEC[11]). Un centro tecnológico creado en 2008 y gestionado por la Fundación Andaluza para el Desarrollo Aeroespacial (FADA). Este centro, desde sus inicios, viene desarrollando una importante labor de investigación y desarrollo de tecnologías relacionadas con el sector que le da nombre, teniendo en la actualidad más de 60 proyectos de I+D+i en marcha.

Las instalaciones de CATEC constan de un edificio de 4.500 m^2 con 3.000 m^2 dedicados a laboratorios y talleres, y 1.500 m^2 de oficinas. Dichas instalaciones están localizadas en el Parque Tecnológico Aerópolis, en San José de la Rinconada, Sevilla.

Figura 2-5. Logos de FADA-CATEC.

El Centro cuenta actualmente con una plantilla de más de 60 empleados, en su mayoría investigadores. CATEC además lidera la investigación, gracias a las sinergias que sus titulados, principalmente ingenieros de diferente índole (Aeronáutica, Industrial, de Telecomunicaciones, Mecatrónica, de Materiales, Química o Informática) confieren al centro.

Los Objetivos tecnológicos del Centro son:

(i) desarrollar las Líneas Tecnológicas definidas en el Plan Estratégico: nuevos materiales aeroespaciales; estudio y aplicación de las tecnologías de inspección y ensayos no destructivos; mejora de los procesos de fabricación avanzada; automatización y optimización de procesos industriales; desarrollo de nuevos sistemas y/o subsistemas embarcados; ensayos y certificación de sistemas y subsistemas embarcados; y desarrollo de UAVs y sus tecnologías asociadas,

(ii) el acercamiento de la industria aeroespacial a la fábrica del futuro,

(iii) promover la Cultura de la Innovación, prestando especial interés a las PYMES,

(iv) implantar y mejorar la Gestión del Conocimiento, y

(v) fomentar la Cooperación entre empresas y Centros de Investigación, a nivel nacional e internacional.

Con estos objetivos, el centro, contribuye, de forma destacada, a la mejora de la competitividad de las empresas del sector mediante el desarrollo de proyectos de I+D+i, el impulso de la creación de conocimiento, la gestión de la propiedad intelectual y la innovación tecnológica. Todo este trabajo permite mejorar y dar respuestas cada vez más innovadoras a los problemas del sector aeroespacial, a la vez que se anticipa con visión de futuro a las nuevas necesidades.

[11] http://www.catec.aero/en

CATEC cuenta con una importante experiencia en el desarrollo y aplicación de nuevas tecnologías TIC para mejorar la eficiencia en los procesos de fabricación aeronáuticos. El Centro desarrolla soluciones innovadoras para sistemas de fabricación, ensayos y verificación, integrando sistemas de información heterogéneos, desplegando escenarios colaborativos avanzados entre personas y robots con simulación y programación avanzada, y explotando tecnologías relacionadas con la fusión sensorial, la inteligencia artificial y la realidad virtual y aumentada. Asimismo, fruto del éxito y alcance que ha tenido ROS en el ámbito investigador como consecuencia de su utilidad, versatilidad y escalabilidad, CATEC también hace uso de este software en el desarrollo de estas soluciones.

Muestra de estos desarrollos son los numerosos proyectos de I+D que ha realizado para la Industria Aeronáutica junto a empresas como Airbus, Airbus D&S, Aernnova, Alestis, Boeing, Cesa, CT Ingenieros, DLR o Safran, entre los que se encuentran:

- ROMEO: desarrollo de un sistema robótico flexible que integra un efector final para avellanado y achaflanado de piezas de fibra de carbono y que permite mecanizar núcleos.
- FUTURASSY: aplicación de un sistema robótico industrial para uso en montajes estructurales de fibra de carbono en la industria aeronáutica.
- ODISEO: desarrollo e investigación en sistemas eléctricos. Se desarrolla un sistema de realidad aumentada para ayudar al operario en la fabricación de mazos eléctricos usando una tecnología de identificación a través de marcadores para la toma de decisiones a la hora de ejecutar acciones, además de proyección de recursos sobre la mesa de trabajo del operario.
- VECTURA: desarrollo de nuevas tecnologías de montaje y pintura de aeroestructuras para la fabricación de helicópteros y otras estructuras ligeras de aeronaves de baja cadencia. Robotización de los procesos de taladrado y aplicación de pintura, y desarrollo de un sistema de proyección láser para la señalización del flujo de trabajo y defectos superficiales en el proceso de inspección.
- DIANNA: desarrollo e investigación de nuevas tecnologías para la automatización de los procesos de montaje aeronáutico como por ejemplo el desarrollo de un robot para la inspección de espacios estrechos, que es capaz de llegar a zonas de alto confinamiento, ahorrando tiempo de operación y previniendo riesgos para los trabajadores
- SACA: desarrollo de sistemas electrónicos de ayuda a la integración de mazos de cables y conectores. Se usa un sistema de proyección de luz natural para ayudar al operario en el enrutamiento de mazos de cables a lo largo de la estructura del avión.
- SINGULAR: diseño de algoritmos para la identificación automatizada de piezas empleando machine learning y así como el desarrollo hardware de una estantería inteligente para el almacenamiento y clasificación de componentes metálicos.
- CEMTAURO: desarrollo de una celda de trabajo robótica multiproceso para operaciones aeronáuticas de fabricación y montaje con robots antropomórficos e inspección con tecnologías de realidad aumentada. Estas últimas incluyen un sistema de proyección con luz natural para ayudar a la colocación de etiquetas de identificación sobre los tubos, un sistema de proyección láser para aplicaciones asistencia en los procesos de montaje de conjuntos aeronáuticos, y un sistema automático con luz estructurada para medir la altura de la cabeza de los remaches de los situados en la superficie exterior de la aeronave para evaluar el borde correcto con la superficie.
- AEROPAINT: logística y robotización de los procesos de pintura aeronáutica. Se desarrolla un sistema que integra diversas soluciones tecnológicas para lograr un proceso automatizado de pintado y verificación de componentes aeronáuticos con alta productividad, eficiencia y respeto por el medioambiente. Para ello, se aplican técnicas de realidad aumentada para asistencia al operario, robotización de procesos tradicionales, desarrollos para la colaboración humano-robot o técnicas de inteligencia artificial en procesos de inspección por visión artificial, y lograr así un modelo de fábrica interconectada.

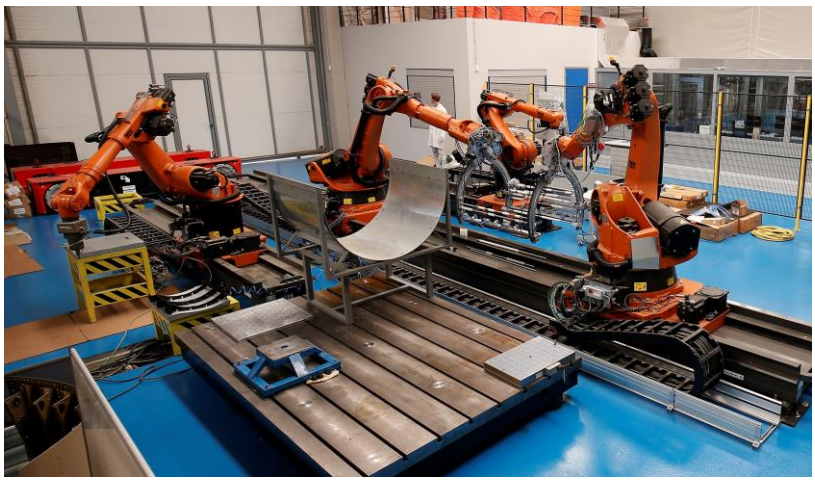


Figura 2-6. Célula robótica en las instalaciones de CATEC.

En este punto y dada su amplia experiencia previa en el uso nuevas tecnologías como es la realidad aumentada, en el caso de proyectos como VECTURA, CEMTAURO o AEROPAINT, CATEC ha querido aprovechar la oportunidad para embarcase en un proyecto dentro de la convocatoria ROSIN, que ha dado pie al desarrollo del presente trabajo.

Una vez desarrollada la memoria técnica y entregada la solicitud, tras ser aprobada en la resolución definitiva de los beneficiarios de la tercera convocatoria de ROSIN, da comienzo el proyecto Z_LASER_PROJECTOR[12], que como ya se ha explicado anteriormente, tiene como objetivo impulsar el desarrollo de aplicaciones de realidad aumentada basadas en ROS, que utilicen esta tecnología para ofrecer soluciones en el ámbito industrial accesibles para las PYMEs.

[12] https://www.rosin-project.eu/ftp/z_laser_projector-a-package-for-a-laser-projector-to-support-with-augmented-reality-regular-industrial-applications

3 ESTADO DEL ARTE

Una de las definiciones más comúnmente aceptada es la proporcionada por el investigador Ron Azuma, el cual dice que la Realidad Aumentada (RA) es una tecnología que cumple con tres requisitos clave [3]:

1) Combina contenido real y virtual.
2) Es interactiva en tiempo real. De alguna manera puede generar gráficos interactivos que respondan a las entradas del usuario en tiempo real.
3) Se presenta en 3D. Dispone de un sistema de seguimiento ('tracking') que permite posicionar y orientar el punto de vista del usuario de manera que la imagen virtual aparezca fija en el mundo real.

En un contexto más amplio, la Realidad Aumentada es una de las últimas tecnologías desarrolladas con el fin de hacer invisibles las interfaces de los ordenadores y mejorar la interacción del usuario con el mundo real.

Un enfoque alternativo es el de la Realidad Virtual (RV). En este caso, la visión del mundo real del usuario es completamente reemplazada por gráficos generados por computador y, de nuevo, el ordenador vuelve a ser invisible. Muchas de las herramientas utilizadas en ambas tecnologías son las mismas, por ejemplo, las pantallas montadas en la cabeza, los sistemas de seguimiento, los dispositivos interactivos manuales, etc. Sin embargo, hay una diferencia importante entre los sistemas de RA y RV. El objetivo principal de un sistema de Realidad Virtual es utilizar la tecnología para reemplazar la realidad, empleando una interfaz de visualización totalmente inmersiva, con un amplio campo de visión (FOV) y unos gráficos 3D tan realistas como sea posible. Este campo de visión hace referencia a la cantidad de espacio visual del usuario que es ocupado por gráficos generados por el ordenador. Cuanto mayor es el FOV, mayor es el nivel de inmersión, menor es la cantidad del mundo real que puede ver el usuario, y menor es, también, el nivel de precisión del sistema de RV necesario para el seguimiento del punto de vista con respecto al mundo real.

Por el contrario, el objetivo principal de un sistema de Realidad Aumentada es complementar y mejorar las interacciones con el mundo real mediante contenido digital de una forma no inmersiva, con un FOV más pequeño y unos gráficos 3D menos complejos. En este caso, el seguimiento sí que debe ser lo más preciso posible para crear la ilusión de que el contenido virtual existe verdaderamente en el mundo real. Por ejemplo, en una interfaz de RA transparente es muy fácil para el ojo humano percibir un desajuste de unos pocos milímetros entre gráficos reales y virtuales.

Otra forma de definir la Realidad Aumentada es en el contexto de otras tecnologías. Milgram y Kishino [4] introdujeron el concepto de "Realidad Mixta" como la fusión de los mundos real y virtual, y el concepto de Continuo de Realidad Mixta, refiriéndose a una taxonomía de las diversas formas en que los elementos "virtuales" y "reales" pueden combinarse entre sí. En el extremo derecho de este continuo, se encuentra el Entorno Virtual, en el que la visión del mundo del usuario se sustituye completamente por el contenido virtual generado por ordenador. En el extremo opuesto, está el Entorno Real donde la visión del usuario no se reemplaza por contenido virtual. Cerca de estos extremos, próximo al Entorno Virtual, está la Virtualidad Aumentada (VA), donde la mayor parte de la vista del usuario es reemplazada por gráficos 3D, pero sigue existiendo una parte de la visión del mundo real disponible, mientras que cerca del extremo del Entorno Real, está la Realidad Aumentada, donde los gráficos virtuales mejoran la vista del usuario del mundo real.

De esta manera, a medida que se añade más o menos contenido virtual al Entorno Real, la interfaz se acerca o se aleja de la escena del Entorno Virtual. La principal conclusión de esta taxonomía es que las interfaces de Realidad Aumentada no se sitúan en un punto discreto entre las experiencias reales y las virtuales, sino que pueden aparecer en cualquier lugar a lo largo del Continuo de la Realidad Mixta[5].

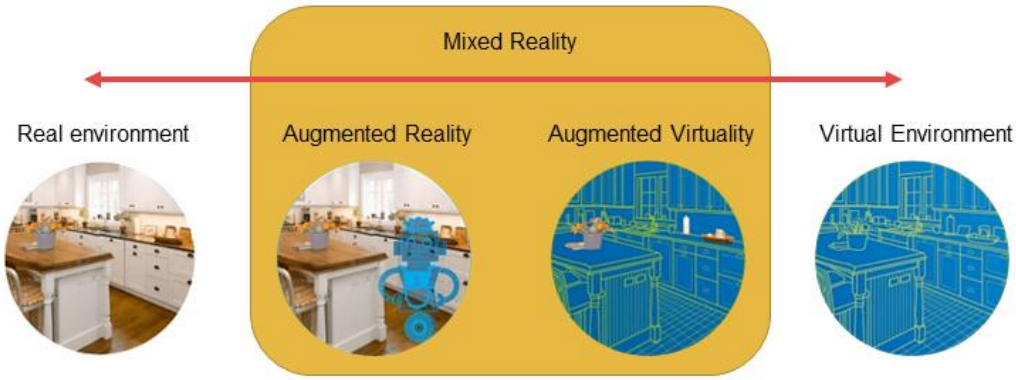


Figura 3-1. Continuo de Virtualidad.

3.1 Tecnologías

La RA no se restringe a un tipo particular de tecnologías como las pantallas montadas en la cabeza (HMD), ni si quiera se limita únicamente al sentido de la vista[6], sino que puede aplicarse potencialmente para aumentar otros sentidos como el oído. En el marco de este trabajo se enfoca la investigación en las tecnologías relacionadas con la visión[7] de las que se pueden distinguir principalmente dos tipos:

- Sistemas de realidad aumentada basados en tecnologías de proyección.

Estos sistemas se caracterizan por emplear un proyector para mostrar la información sobre una región determinada utilizando para ello diferentes tipos de fuentes de generación de luz. Por un lado, se encuentran los sistemas de proyección tradicionales, que utilizan una lámpara como fuente luminosa. Estos sistemas son mucho más baratos y permiten proyectar una gran variedad de información, desde imágenes con detalles geométricos, hasta información textual, fotos y videos en color con las explicaciones de las operaciones a realizar. Por contrapartida, estos proyectores están mucho más expuestos a los ruidos de la luz, lo que los hace casi inútiles en ambientes bien iluminados.

Estos sistemas han ido evolucionando considerablemente hasta llegar a unos niveles de calidad y precisión muy exigentes. 3LCD, DLP y LCoS, son algunas de las tecnologías más desarrolladas hasta la fecha. De ellas, la más extendida en la actualidad es 3LCD[8] que utiliza una fuente de luz blanca descompuesta en los tres colores primarios utilizando una combinación de espejos. El panel LCD propio de cada color recibe las señales eléctricas correspondientes a la imagen y, finalmente, estas imágenes son combinadas mediante un prisma que genera la imagen final en color que será proyectada a través de una lente. Serán la resolución del dispositivo y la calidad de la lente las que determinen en última instancia la precisión del sistema.

Figura 3-2. Aplicación de realidad aumentada basada en proyección tradicional para ayudar al operario en el proceso de ensamblaje.

Por otro lado, están los proyectores que utilizan fuentes de luz láser, cuyas principales ventajas son la precisión y el excelente contraste que presentan, una propiedad muy útil en ambientes industriales con gran ruido

luminoso. Esta tecnología permite proyectar, utilizando un rayo láser, algunos detalles geométricos como puntos, líneas, formas o incluso información textual. Generalmente, el color utilizado en esos proyectores es el verde (532nm de longitud de onda), porque es el que mejor capta el ojo humano.

El principio por el que funcionan estos proyectores consiste en una fuente que produce un punto láser, esta se enfoca y se dirige mediante espejos. Este movimiento del punto a lo largo de una serie de posiciones definidas de una línea o curva es lo que genera la proyección de un contorno. El movimiento del punto láser solo tiene que ser lo suficientemente rápido como para producir una imagen que el ojo humano percibe como una figura. El inconveniente que presenta esta tecnología es que cundo se pretende mostrar una gran cantidad de información al mismo tiempo, la frecuencia de proyección no es la suficiente y la información parpadea sobre la superficie.

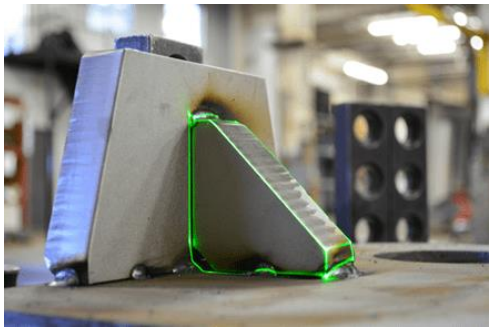


Figura 3-3. Aplicación de realidad aumentada basada en proyección láser para ayudar al operario en el proceso de mecanizado.

- Sistemas de realidad aumentada basados en pantallas.

Son sistemas donde la información no se proyecta directamente sobre la superficie de trabajo, sino que se muestra en una interfaz intermedia[9]. Una de las tecnologías más extendidas son las basadas en la pantalla de dispositivos portátiles como smartphones o tablets (VST, 'Video See-Through'). En estos dispositivos, la cámara de vídeo integrada muestra en tiempo real las imágenes captadas sobre las que se superpone la información adicional de carácter virtual. Por lo general, se utiliza un conjunto de marcadores para detectar la posición de los objetos virtuales en el entorno real. De este modo, el sistema de posicionamiento localiza constantemente estos marcadores con la mayor frecuencia posible y la pantalla refresca la imagen. Las precisiones medias de este tipo de sistemas son del orden del milímetro.

Dado que el procesamiento de video se realiza antes de mostrar el contenido al usuario, es posible ajustar las opciones de imagen como el brillo o el contraste del mundo real. Sin embargo, existen algunos inconvenientes entre los que se incluyen la baja resolución de la realidad, un campo de visión limitado y el paralaje ('parallax') ocular producido por la distancia que existe entre la ubicación exacta del ojo del usuario y la posición de la cámara.

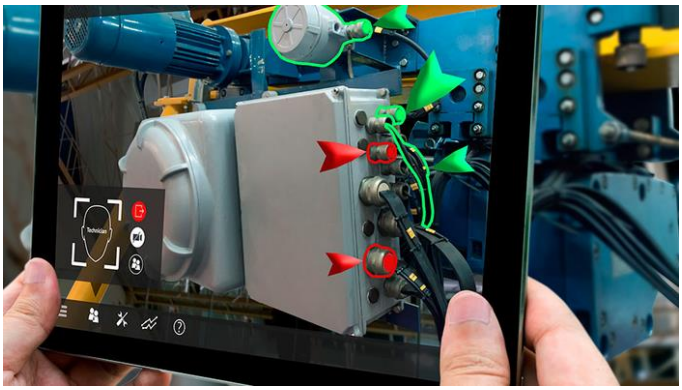


Figura 3-4. Aplicación de realidad aumentada basada en VST.

Una alternativa a los sistemas que trabajan con el vídeo capturado por las cámaras integradas en estos dispositivos portátiles, son las pantallas transparentes (OST, 'Optical See-Trough') basadas en la tecnología

AMOLEO, muy similar a la conocida tecnología OLED. Este tipo de pantallas transparentes dejan pasar una cantidad suficiente de luz del mundo real lo que permite ver el entorno directamente, al mismo tiempo que se proyectan sobre ellas las imágenes generadas por ordenador mediante un proceso de electrofosforescencia, a través de un componente de visualización colocado encima o en el lateral.

Ejemplo de esto son los visores o gafas de realidad aumentada, cuya ventaja principal es dejar libres las manos del usuario para facilitar la ejecución de las tareas. Además, estos sistemas, como permiten ver directamente, muestran el mundo en su resolución real evitando los posibles desajustes causados por el desplazamiento de la posición la cámara en relación con el ojo del usuario. Sin embargo, se trata de una tecnología no del todo madura todavía, por lo que presenta muchos defectos asociados con la pixelación de la imagen y la velocidad de actualización, especialmente en los movimientos bruscos. Estos defectos, además, al representar la imagen tan cerca del ojo, producen incomodidades en el usuario.

Figura 3-5. Aplicación de realidad aumentada basada en OST.

A continuación, se enumeran en una tabla las principales ventajas y desventajas de las tecnologías mencionadas:

Tabla 3–1. Ventajas y desventajas de las tecnologías de RA.

Tecnología		Ventajas	Desventajas
Basada en proyección	Fuente de luz LED	- Más barata - Más flexible - Recambios más fáciles	- Muy afectada por ruido lumínico
	Fuente de luz laser	- Más precisa - Excelente contraste - Menor consumo	- Más cara
Basada en pantallas	VST	- Esencial para aplicaciones remotas - Opciones de vídeo (brillo, filtros de imagen)	- Depende de baterías - Dificulta el trabajo manual - Baja resolución de la realidad y FOV limitado
	OST	- Visión directa del mundo real (resolución real) - Facilita el trabajo manual	- Depende de baterías - Molestias en los usuarios

3.2 Aplicaciones

Como se ha dicho, la Realidad Aumentada mejora la percepción e interacción del usuario con el mundo real a través de la información transmitida por los objetos virtuales que el usuario no puede detectar directamente con sus propios sentidos, ayudándolo a realizar sus tareas. Estas ventajas han extendido el uso de esta tecnología a numerosos sectores, entre los que se encuentran:

- Medicina: este campo toma su principal motivación en la necesidad de visualizar los datos médicos y el paciente dentro del mismo espacio físico, lo requiere la visualización in situ y en tiempo real de datos heterogéneos registrados conjuntamente[10]. Es el caso de la RA quirúrgica, que proporciona a los médicos orientación, ayuda y apoyo con información valiosa durante una operación, o que también puede servir como apoyo en el ensayo o la discusión de la operación, utilizando una versión virtual realista del órgano del paciente antes de la operación quirúrgica real. Al mismo tiempo, la formación quirúrgica basada en la AR puede mejorar la experiencia de los cirujanos en las operaciones en vivo mediante funciones que permitan mostrar información como los diagnósticos del paciente, las imágenes radiológicas y los datos fisiológicos. Gracias al avance de la tecnología en el campo del modelado y la simulación por computador, se ha aumentado la calidad y el valor del paciente virtual en 3D que se crea de la cabeza a los pies utilizando imágenes de resonancia magnética y tomografía computarizada, de manera que los médicos pueden ahora inspeccionar el cuerpo del paciente desde cualquier ángulo sin tener que realizar un procedimiento quirúrgico, permitiéndoles formular un diagnóstico más exacto mientras se minimiza el impacto físico. Otro ejemplo del uso de esta tecnología es el sistema de visión de venas: un escáner de mano que se proyecta sobre la piel para mostrar la ubicación de las venas del paciente, aumentando en 3,5 veces las probabilidades del médico de encontrar con precisión una vena durante el primer intento de inyección. Otro ejemplo más doméstico son los relojes inteligentes que son capaces de medir y llevar un seguimiento de las constantes vitales y la calidad del sueño, con el fin de facilitar el cuidado de la salud.
- Entretenimiento: la RA se introdujo por primera vez en la industria de la televisión a través de la previsión meteorológica para la que se utilizaron símbolos gráficos en 3D y tablas meteorológicas. Desde entonces, numerosos han sido los usos de esta tecnología en otros géneros televisivos, como, por ejemplo, en la radiodifusión de eventos deportivos para señalar la ubicación de un disco de hockey difícil de ver cuando se mueve rápidamente por el hielo, para mostrar la información de los pilotos de carreras, para resaltar la trayectoria de la pelota de golf sobre el campo, etc. La realidad aumentada se aplica también en otras áreas de la industria del entretenimiento como son la música y el teatro incluyendo marcadores de realidad aumentada en el arte de las fundas de vinilo con los que los autores pueden proporcionar todo tipo de información adicional, desde la letra hasta la creación del álbum, o con el uso de gafas de realidad aumentada para mostrar los subtítulos de la obra de teatro directamente en el fondo del escenario. Por último, la tecnología AR también expande los entornos de los videojuegos al mundo real. Recientemente, se han implementado juegos con RA en los teléfonos inteligentes, como es el caso del exitoso juego "Pokémon GO" en el que se utiliza la tecnología GPS para el seguimiento de la ubicación del usuario.
- Educación: las nuevas posibilidades de enseñanza y aprendizaje que ofrece la RA han sido cada vez más reconocidas por los docentes. La coexistencia de objetos virtuales y entornos reales permite a los alumnos visualizar relaciones espaciales complejas y conceptos abstractos, experimentar fenómenos que no son posibles en el mundo real, interactuar con formas geométricas tridimensionales, y desarrollar importantes prácticas que no pueden desarrollarse y promulgarse en otros entornos de aprendizaje[11]. Estos beneficios educativos han hecho de la RA una de las tecnologías emergentes para la educación en los próximos años.
- Marketing: la RA se ha convertido en una de las tendencias publicitarias más importantes en términos de comercio, usándose por primera vez para el marketing en la industria automotriz. Algunas compañías imprimieron volantes especiales que eran reconocidos automáticamente por unas cámaras, haciendo que se mostrara en la pantalla el modelo tridimensional del coche anunciado. Este enfoque se extendió luego a varios nichos de mercado, desde anuncios comerciales de televisión en 3D durante los partidos de fútbol, hasta productos de consumo como ropa[12] o muebles. En estos últimos, la RA se utiliza para mejorar las previsualizaciones de los productos en 3D creados virtualmente, permitiendo una manipulación interactiva de los productos sin abrir. El código QR es otro ejemplo

muy simple del uso de esta tecnología en este sector: el código en blanco y negro se convierte en información más compleja cuando es escaneado por un teléfono móvil o una computadora. Durante la situación de emergencia causada por el coronavirus el código QR se ha usado de manera muy extendida, por ejemplo, para el escaneo del menú de los restaurantes.

- Turismo: la tecnología de RA se utiliza en este sector para tratar de mejorar la experiencia durante la visita a sitios de interés cultural como exposiciones, museos, etc., mediante la combinación de diferentes técnicas como, por ejemplo, las audioguías, la recreación virtual de escenografías de la época, guías multimedia en el móvil, etc. Otra aplicación de realidad aumentada es el uso de los teléfonos inteligentes como traductor de un idioma extranjero permitiendo mantener una conversación en tiempo real.
- Fabricación: la investigación sobre las aplicaciones de la RA en la fabricación es un área fuerte y en crecimiento. El desafío es diseñar e implementar sistemas integrados de fabricación de RA que puedan mejorar los procesos de fabricación, así como el desarrollo de productos y procesos, lo que conduce a un plazo de entrega más corto, a un coste reducido y a una mejor calidad. Algunos ejemplos de aplicaciones de RA enfocadas a este sector se desarrollan en el siguiente apartado.

3.3 Proyección láser aplicada a la fabricación

En el actual entorno empresarial altamente competitivo y dinámico, la industria manufacturera se enfrenta a nuevos retos que requieren una perspectiva holística de los cuatro pilares principales de la fabricación: el coste, el tiempo, la calidad y la flexibilidad. Las empresas manufactureras deben producir productos innovadores a bajo costo con un tiempo de comercialización reducido. La alta mezcla de productos con bajo volumen, la personalización para satisfacer las demandas individuales de los clientes, la creciente legislación sobre cuestiones ambientales y de otro tipo, han hecho que los procesos de fabricación sean más complejos y exigentes. Además, la tendencia creciente de los entornos de fabricación globalizados requiere intercambios de información en tiempo real entre las diversas unidades funcionales del ciclo de vida de desarrollo de un producto: el diseño, la planificación, la programación de la producción, el mecanizado, el montaje, etc., así como una tarea fluida de colaboración entre ellas, obligando a los procesos de fabricación a ser más receptivos y sistemáticos para ser eficientes y económicamente competitivos.

Debido a los recientes avances en las tecnologías de la información y la comunicación (TIC), la investigación sobre la fabricación ha entrado en un nuevo nivel de planificación de productos, análisis y solución de problemas. La fabricación digital ha sido considerada, durante la última década, como un conjunto de tecnologías muy prometedoras para reducir los tiempos y los costos de desarrollo de los productos, así como para abordar la necesidad de personalización, el aumento de la calidad de los productos y una respuesta más rápida al mercado.

Esta digitalización se ha convertido en una tendencia común en todo el mundo, en la que los sistemas de fabricación integrados por ordenador se aplican a cualquiera de las fases del proceso de fabricación eliminando los errores de manejo de datos y mejorando la toma de decisiones, ofreciendo así la oportunidad de reducir los tiempos y los costes. Entre estas fases, la de montaje es una de las más comunes en el proceso de fabricación de casi cualquier producto. Esta tarea está formada por diferentes pasos en los que los distintos componentes se ensamblan, de forma automatizada o mediante el uso de herramientas por parte de los operarios, para dar como resultado el producto final.

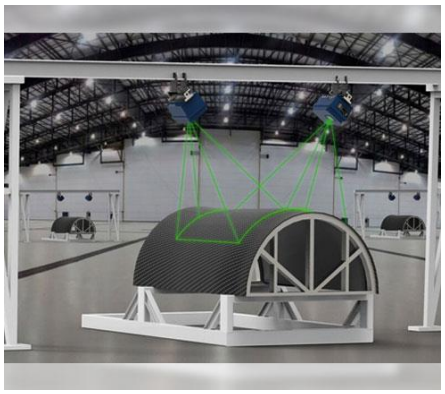
Entre las tecnologías utilizadas como medio hacia la fabricación inteligente, la tecnología de realidad aumentada (RA) ha madurado y ha demostrado ser una solución innovadora y eficaz, capaz de proporcionar, en combinación con las capacidades humanas, herramientas eficientes y complementarias para ayudar a resolver algunos de los problemas críticos con el fin de mejorar los procesos de fabricación[13].

Un ejemplo de aplicación de realidad aumentada al proceso de fabricación es el uso de tecnología de proyección láser como sustitución a las plantillas físicas utilizadas por los operarios como guías en las tareas de montaje. El objetivo es crear un sistema de guiado que mejore la comprensión de estas tareas proyectando instrucciones y secuencias de animación que se pueden precodificar en la etapa de diseño. Estas secuencias se superponen sobre los productos reales en las líneas de montaje a medida que se necesitan. De esta manera, se consigue:

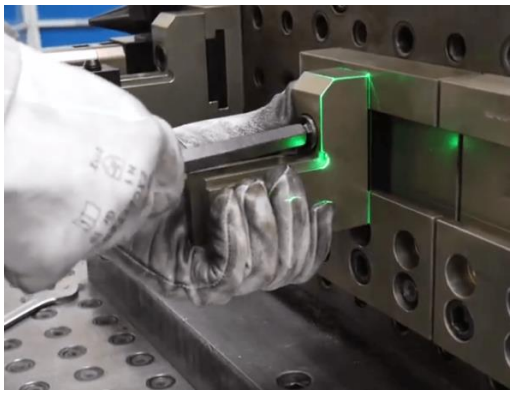
- Mejorar las tolerancias y aumentar la calidad del producto gracias a la precisión que ofrecen estos sistemas, evitando los errores humanos en el ajuste de plantillas, que en ocasiones son sometidos a trabajos rutinarios, estresantes, poco ergonómicos o que conllevan determinadas cargas de fatiga visual.
- La formación necesaria para los operarios es menor.
- Aumentar el rendimiento y la productividad del proceso, ya que las instrucciones y animaciones se pueden actualizar de forma más rápida y cómoda que las plantillas físicas.
- Disminuir los residuos generados por la fabricación de plantillas que en algunos sectores pueden llegar a ser incluso de titanio de varios centímetros de grosor.

A continuación, se exponen algunos casos de uso:

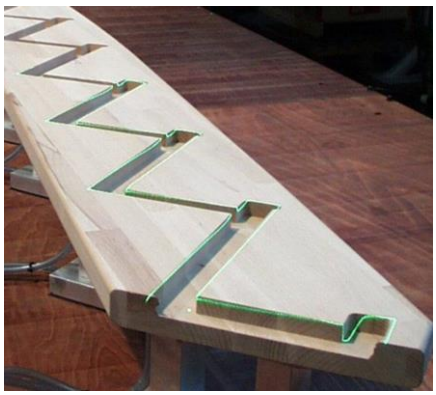
- En la industria aeronáutica se utilizan proyectores láser para la colocación de materiales. Las líneas proyectadas ayudan a inspeccionar visualmente si las piezas y subconjuntos están en su lugar; ayudando a verificar la alineación óptima de las piezas de trabajo y comprobar la precisión dimensional no crítica, a simple vista[14]. Además, los proyectores láser se utilizan hoy en día para guiar a los usuarios mientras limpian o pulen grandes superficies de aeronaves, mostrándoles los caminos a seguir y las áreas completadas[15].
- En la producción de automóviles[16][17], los proyectores láser se utilizan para posicionar elementos de ensamblaje como cierres y soporte[18]. Su pleno potencial también se despliega en la industria cuando el proyector muestra los mazos de cables eléctricos y los planos de conexión; incrementando la velocidad en muchas tareas de cableado.
- En la industria de materias primas como la madera, la proyección láser es ideal para aplicaciones en las que el material de trabajo debe alinearse manualmente con las hojas de sierra, trayectorias de fresado o puntos de taladrado.
- En el sector de la logística también se utilizan proyectores láser para ayudar a los trabajadores de los almacenes a localizar los productos y/o cumplir los pedidos. Su uso ha demostrado que mejora la precisión y la rentabilidad general.



(a)



(b)



(c)

(d)

Figura 3-6. Ejemplos de uso de proyección láser en procesos industriales: a) en el proceso de mecanizado de aeronaves, b) proceso de montaje de soportes en la fabricación de vehículos, c) perfil de corte de pieza de madera, d) posicionamiento de pedidos sobre el carro de transporte.

4 HERRAMIENTAS Y EQUIPOS

En este apartado se describen las principales características técnicas del sistema de proyección de láser así como la herramienta software utilizada para implementar la comunicación y control del dispositivo desde el ordenador.

4.1 Modelo ZLP1

Como se ha comentado anteriormente, el ZLP1 es el modelo de proyector láser más pequeño de la serie ZLP de la marca Z-LASER. Se trata de un dispositivo seguro para los ojos (clase 2M) que se utiliza para proyectar objetos y figuras a modo de guía óptica para las tareas de los operarios en los procesos de producción industrial. Está optimizado para aplicaciones 2D y 3D, tratando de ampliar y optimizar la producción y el flujo de operación en las estaciones de trabajo, en logística, pick-and-place, control de calidad, etc.

Figura 4-1. Modelo proyector láser ZLP1 de la marca Z-LASER.

A continuación, se describen en una tabla los principales detalles técnicos extraídos del 'datasheet'[19] del fabricante:

Tabla 4–1. Especificaciones técnicas del proyector láser ZLP1.

Tecnología	Diodo láser
Longitud de onda	520 nm (color verde)
Potencia de salida	4 mW
Voltaje de operación	24 VDC ± 10%
Consumo de potencia típico	40 W - 70 W
Peso	3.4 kg
Dimensiones (L x W x H)	314 x 111 x 95 mm
Frecuencia de proyección	Máx. 50 Hz (depende de la proyección)
Ángulo de apertura del haz	Máx. 60º x 60º. Se recomienda 56º
Distancia de proyección	De 1.0 m a 3.0 m
Área de proyección	De 1.0 m x 1.0 m a 3.5 m x 3.5 m

Precisión	± 3 mm/m (depende de la distancia)
Tipo de láser (estándar EN 60825-1:2014)	2M
Grado de protección IP (estándar EN 60529)	IP54
Comunicación	Ethernet
Sistema de enfriamiento	Activo o pasivo

De los detalles descritos en la tabla anterior, se observa que el dispositivo se clasifica como un proyector láser de clase 2M. Esta clasificación, recogida en la Norma IEC 60825-1:2014: "*Seguridad de los productos láser. Parte 1: Clasificación de los equipos y requisitos.",* hace referencia al nivel de emisión accesible en comparación con el límite de emisión accesible (LEA) que determina el nivel de peligrosidad para cada clase. Atendiendo a este criterio, los dispositivos láser se clasifican en las siguientes categorías:

- Clase 1: El rayo láser es inofensivo.
- Clase 1M: El rayo láser es inofensivo a menos que se utilicen instrumentos ópticos como lupas o prismáticos.
- Clase 2: El rayo láser está en el rango visible y es inofensivo en el corto plazo (hasta 0.25 segundos).
- Clase 2M: Como la clase 2 mientras no se utilicen instrumentos ópticos como lupas o prismáticos.
- Clase 3R: El rayo láser disponible es peligroso para el ojo.
- Clase 3B: El rayo láser es peligroso para el ojo, e incluso para la piel.
- Clase 4: El rayo láser es muy peligroso para el ojo y peligroso para la piel. La radiación dispersa difusa también puede ser peligrosa. La potencia del láser puede causar un peligro de incendio o explosión.

A la vista de la clasificación, es necesario asegurarse que las personas que utilizan un dispositivo láser de clase 2 a clase 4 hayan sido instruidas sobre el modo de operación, así como estar provistas de los Equipos de Protección Individual necesarios para operar el dispositivo, como es el caso de gafas de seguridad ante radiación láser. Así, también, si el láser utilizado es de la clase 2 o superior y el rayo invade otras áreas de trabajo, se debe asegurar que el alcance del láser esté marcado de forma clara y permanente de acuerdo con las disposiciones legales.

Finalmente, añadir, respecto a la especificación del área de proyección del proyector, que esta depende de la distancia de proyección o, lo que es lo mismo, de la distancia entre el dispositivo y la superficie de proyección y del ángulo de apertura del aparato. Para calcular la distancia de instalación ideal de manera que abarque completamente la superficie de proyección deseada, se puede hacer uso de la siguiente fórmula:

$$h = f * \frac{0.5}{\tan\left(\frac{\alpha}{2}\right)} \tag{4–1}$$

en la que,

- h es la distancia de proyección,
- f es el ancho de la superficie de proyección, y
- α es el ángulo de apertura (se recomienda 56º).

Por ejemplo, para un área de proyección deseada de 2000 x 2000 mm y un ángulo de apertura de 56º, se tiene una distancia de instalación de 1880,7 mm, como se puede ver en la siguiente imagen.

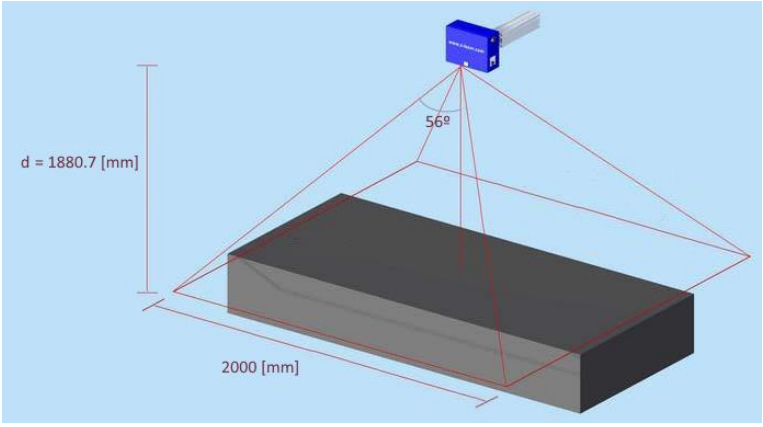


Figura 4-2. Representación de las dimensiones características en un sistema de proyección láser.

De manera adicional, se incluye la siguiente tabla en la que se muestran algunos valores típicos de distancias de instalación del proyector y las correspondientes áreas de proyección. Se puede observar que el área máxima de trabajo se corresponde con una distancia de instalación máxima de 3 metros.

Tabla 4–2. Área de trabajo en función de la distancia de instalación, con un ángulo de apertura máximo de 60º.

Distancia a la superficie de proyección (en mm)	Área de proyección (en mm^2)
1.000	1.155 x 1.155
1.500	1.732 x 1.732
2.000	2.309 x 2.309
2.500	2.887 x 2.887
3.000	3.464 x 3.464

4.2 Apache Thrift

Se supone la siguiente situación en la que múltiples aplicaciones internas que realizan diferentes tareas han sido desarrolladas por equipos separados en lenguajes diferentes, ¿de qué forma podrían estas aplicaciones comunicarse entre sí? Una posible solución es utilizar varias API REST. Sin embargo, en muchos casos, especialmente cuando se transfieren datos binarios, esta solución no proporciona un rendimiento o capacidad de mantenimiento aceptables. Es el caso que se sucede en este trabajo, por un lado se encuentra el dispositivo de proyección láser que ha sido fabricado y programado por el equipo de una empresa, mientras que, ahora, desde el lado del usuario, se quiere trabajar con el entorno de ROS. Se tienen, entonces, dos aplicaciones independientes (proyector y paquete de ROS) que pretenden comunicarse entre sí.

Como consecuencia de la necesidad de una herramienta que permitiera la comunicación entre aplicaciones, Thrift fue concebido por Facebook en el año 2007[20]. Al poco tiempo, abrieron el código y pasaron el proyecto a la Fundación Apache quien se encarga actualmente del desarrollo[13], convirtiéndose en una herramienta ampliamente utilizada no solo por Facebook, sino por muchas otras empresas como Evernote, Twitter, Netflix, entre otras. Los ingenieros de Facebook continúan trabajando en su propia bifurcación, que existe ahora con el nombre de FBThrift y que, con suerte, se incorporará a Apache Thrift.

El concepto principal de este framework son los servicios, similares a las conocidas clases de los lenguajes de programación orientados a objetos, que se definen de manera sencilla usando los tipos de datos implementados en Apache Thrift, para construir la interfaz entre el cliente y el servidor. Estos tipos de datos son mapeados a sus contrapartes nativas en cada lenguaje, así, por ejemplo, en el caso de un tipo de dato simple como un

[13] https://thrift.apache.org/

entero, este se mapea a un entero en cada lenguaje, mientras que, en el caso de otros más complejos, como un 'set', se convierte, por ejemplo, en un 'array' en PHP o en un 'HashSet' en Java.

Estos servicios son declarados en el denominado documento Apache Thrift (con extensión .thrift) utilizando para ello la sintaxis típica de un Lenguaje de Descripción de Interfaz (IDL[14]), en el que se definen las operaciones, los parámetros y los retornos junto con sus tipos. En el caso del paquete desarrollado en este trabajo, estos servicios están definidos en el archivo 'interface.thrift'.

```
// Ejemplo de archivo .thrift que implementa un servicio para sumar dos números

// namespaces are used in packages generated for each language

namespace php adder
namespace py adder

// you can name your own types and rename the built-in ones.
typedef i32 int

service AddService {

// add method - returns result of sum of two integers
    int add(1:int number1, 2:int number2),
}
```

A partir de este archivo se usa el compilador de Apache Thrift, para generar los 'stubs' cliente/servidor en los diferentes lenguajes[21]. Estos 'stubs' o fragmentos de código, utilizan la librería de Apache Thrift y se encargan de la conversión (mapeo) de los parámetros intercambiados entre el cliente y el servidor durante una llamada a procedimiento remoto (RPC).

Dado que el cliente y el servidor usan espacios de direcciones diferentes, y por tanto los punteros a los parámetros usados en una llamada a función (procedimiento) apuntarían a datos diferentes en un sistema y en otro, la idea principal de estas RPC es permitir que el sistema local (cliente) pueda llamar a procedimientos del sistema remoto (servidor) de tal manera que las máquinas de cada extremo del enlace pueden estar utilizando sistemas operativos y lenguajes informáticos diferentes. En el caso de este trabajo, el ordenador del usuario hace la función de cliente (desarrollado en Python) que llama de forma remota a los procedimientos del sistema de proyección láser.

Gracias a la conversión de los parámetros por los 'stubs', una llamada a un procedimiento remoto parece una llamada a función local para el sistema remoto.

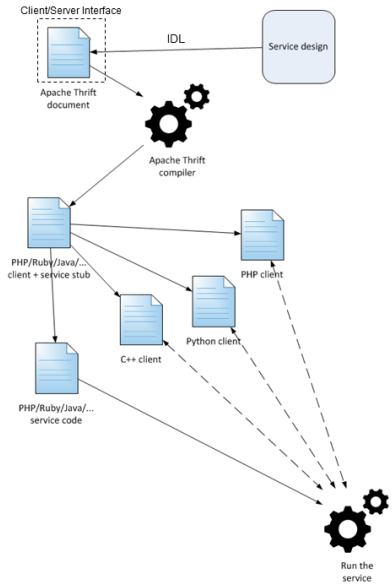


Figura 4-3. Flujo de operación para la programación de una interfaz basada en Apache Thrift.

[14] Un IDL es un lenguaje de especificación utilizado para describir la interfaz de programación de aplicaciones (API) de un componente software de forma independiente al lenguaje, permitiendo la comunicación entre componentes software diferentes.

En conclusión, Apache Thrift ofrece un marco de llamadas a procedimientos remotos y combina una pila de software con un motor de generación de código para construir servicios eficientes y multiplataforma que pueden conectar aplicaciones escritas en una variedad de lenguajes y marcos. Los lenguajes soportados en cualquier combinación de cliente y servidor son: ActionScript, C, C++, C#, Cappuccino, Cocoa, Delphi, Erlang, Go, Haskell, Java, JavaScript, Objective-C, OCaml, Perl, PHP, Python, Ruby, Elixir, Rust, Smalltalk y Swift.

Las diferentes capas que conforman la arquitectura de comunicación de Apache Thrift se pueden ilustrar en la siguiente figura:

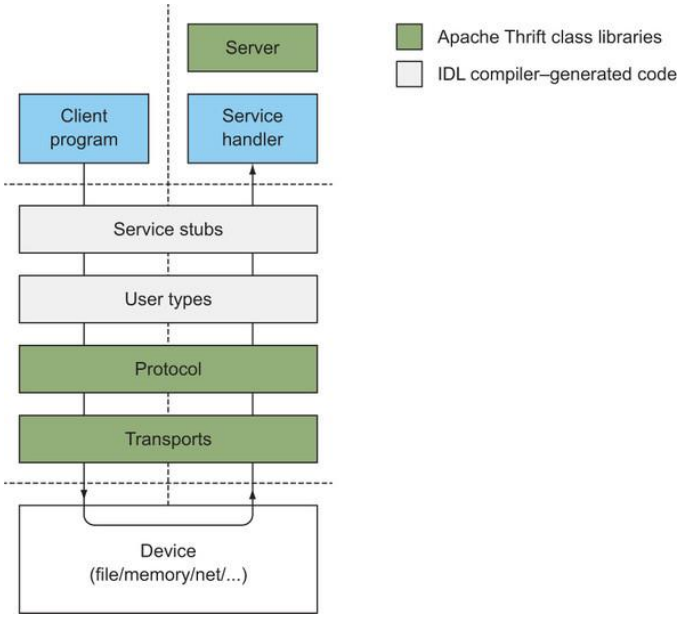


Figura 4-4. Arquitectura de comunicación de Apache Thrift.

Apache Thrift proporciona una biblioteca para definir la capa de transportes, protocolos y servidores. Para esta primera capa de transporte, encargada de efectuar el transporte de los datos de la máquina origen a la de destino independientemente del tipo de red física que esté utilizando, los tipos soportados son:

- TSimpleFileTransport - Este transporte escribe a un archivo.
- TFramedTransport - Este transporte es necesario cuando se utiliza un servidor sin bloqueo. Envía los datos en marcos, donde cada marco es precedido por información de longitud.
- TMemoryTransport - Utiliza la memoria para E/S. La implementación de Java utiliza un simple ByteArrayOutputStream internamente.
- TSocket - Utiliza un socket bloqueante de E/S para el transporte.
- TZlibTransport - Realiza la compresión usando zlib. Se usa en conjunto con otro transporte.

A continuación, se encuentra la capa de protocolo, que es la responsable de codificar y decodificar los datos para que puedan ser transmitidos. Entre los protocolos más populares se pueden encontrar:

- TBinaryProtocol - Un formato binario sencillo, simple, pero no optimizado para la eficiencia de espacio. Más rápido de procesar que el protocolo de texto, pero más difícil de depurar.
- TCompactProtocol - Formato binario más compacto; típicamente más eficiente para procesar también.
- TJSONProtocolo - Utiliza JSON para la codificación de datos.
- TSimpleJSONProtocol - Un protocolo de sólo escritura que no puede ser analizado por Thrift porque deja caer metadatos usando JSON. Adecuado para ser analizado por lenguajes de escritura.

Cuando desee que dos aplicaciones se comuniquen entre sí, debe usar el mismo conjunto de transporte y protocolo para codificar y decodificar la información.

Por último, Thrift incluye una infraestructura de servidores para unir protocolos y transportes, como:

- TNonblockingServer - Un servidor multihilo que utiliza E/S sin bloqueo. TFramedTransport debe ser usado con este servidor.
- TSimpleServer - Un servidor de un solo hilo que utiliza E/S de bloqueo estándar. Útil para pruebas.
- TThreadedServer - Un servidor multihilo usando un modelo de hilo por conexión y E/S de bloqueo estándar.
- TThreadPoolServer - Un servidor multihilo usando una pool de hilos y E/S de bloqueo estándar.

Para terminar, se muestra un esquema de la arquitectura completa de comunicaciones entre los sistemas empleados en este trabajo:

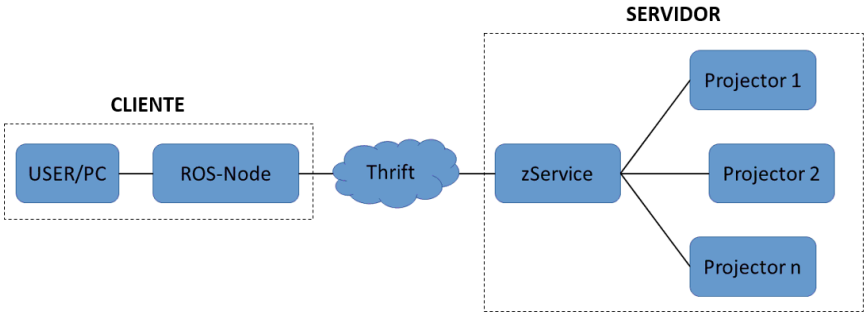


Figura 4-5. Arquitectura de comunicación del sistema de proyección Z-LASER.

donde, como se ha dicho, la aplicación que se ejecuta en el PC del usuario actúa como cliente, que llama de forma remota a los procedimientos (operaciones, transferir datos de proyección, solicitar información de estado, etc.) del servidor que se encuentra en el sistema de proyección láser. Los parámetros utilizados para establecer la comunicación son: TSocket basado en TCP/IP, para el transporte; TBinaryProtocol, para el protocolo; y el servidor TSimpleServer.

5 DESARROLLO

Este apartado está dedicado a realizar una descripción más detallada del paquete software, "Z_LASER_PROJECTOR", desarrollado para el sistema de proyección láser mencionado en los apartados anteriores.

En esta dirección, el siguiente esquema proporciona una visión general de la arquitectura del paquete, en el que se puede comprobar que este está formado a su vez por 4 subpaquetes, cada cual desempeña una función diferente:

- z_laser_msgs: contiene la descripción de los tipos de mensajes y servicios que son comunes y están compartidos por todos los subpaquetes, y que se utilizan para la comunicación.
- z_laser_zlp1: se encarga de la comunicación y control del proyector láser ZLP1 a través de la interfaz de Apache Thrift. En este subpaquete se desarrollan las librerías que recogen las diferentes funcionalidades que ofrece el dispositivo para que sean accesibles desde la API de ROS (topics, servicios, etc.). Entre estas funcionalidades se pueden encontrar: la apertura de la conexión con el proyector, la definición de sistemas de coordenadas, la definición de elementos de proyección, la calibración mediante el escaneo de reflectores, el inicio y paro de la proyección de figuras, el acceso a información almacenada en el sistema de proyección, entre otras, explicadas más adelante.
- z_laser_viz: proporciona un visualizador en "RViz" que simula la proyección de sistemas de coordenadas y elementos definidos por el usuario. Esto ayuda a ajustar la posición, aumentar el detalle de las proyecciones o realizar comprobaciones previas del resultado final.
- z_laser_gui: facilita el uso de la ROS API del paquete a través de una interfaz gráfica de usuario que sintetiza el envío de mensajes a través de los topics, la llamada a los servicios, etc., evitando la necesidad de interactuar con el terminal.

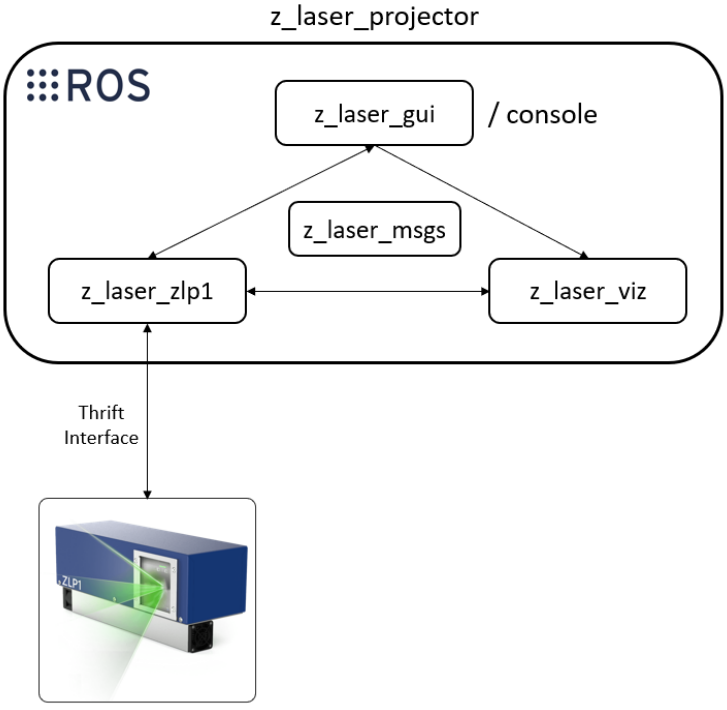


Figura 5-1. Arquitectura del paquete Z_LASER_PROJECTOR.

En resumen y con ayuda del esquema anterior, se puede comprobar que el paquete z_laser_projector se comunica con el proyector mediante la interfaz desarrollada en Apache Thrift a través del subpaquete z_laser_zlp1. Al mismo tiempo, en el marco del ecosistema de ROS, se puede hacer uso de las funcionalidades que ofrece el dispositivo a través del envío de mensajes o la llamada a servicios desde la GUI que proporciona el subpaquete z_laser_gui o directamente desde la consola. Los mensajes son recibidos por los paquetes z_laser_zlp1 y z_laser_viz, que se encargan de ejecutar las tareas correspondientes en el proyector y en el visualizador respectivamente.

Todos los archivos descritos en este apartado se pueden encontrar en el repositorio público del proyecto, en GitHub: https://github.com/fada-catec/z_laser_projector. Así también, la documentación oficial se puede encontrar en la correspondiente página de la Wiki de ROS: http://wiki.ros.org/z_laser_projector

5.1 z_laser_msgs

Como ya se ha comentado, este subpaquete recoge los tipos de mensajes y servicios que son comunes, es decir, que son importados y utilizados por el resto de subpaquetes y que se utilizan para la comunicación. A continuación, se describen estos tipos:

- Figure.msg

Se trata de un tipo de mensaje que recoge todos los parámetros con los que se puede definir cualquiera de los tipos de elementos de proyección básicos admitidos por el proyector: líneas, círculos, arcos, elipses y cadenas de caracteres.

`uint8 POLYLINE=0`
`uint8 CIRCLE=1`
`uint8 ARC=2`
`uint8 OVAL=3`
`uint8 TEXT=4`
	Valores definidos para los tipos de figuras.

`int32 figure_type`	Identificador del tipo de figura. Vale 0 para la línea, 1 para el círculo, 2 para el arco, 3 para la elipse y 4 para la cadena de caracteres
`string projection_group`	Nombre del grupo de proyección al que pertenece el elemento de proyección
`string figure_name`	Nombre del elemento de proyección
`geometry_msgs/Point position`	Posición del punto característico del elemento de proyección. En el caso de la línea y la cadena de texto es el punto inicial, mientras que, para el círculo, el arco y la elipse, es el centro de la figura
`float64[] size`	Vector de dimensiones características. Este vector estará formado por una o dos componentes como máximo y reúne los valores de las dimensiones necesarias para definir el elemento de proyección. En el caso de una línea solo se necesita una componente que será la longitud de esta, mientras que en el caso del círculo y el arco será el radio. Por otro lado, la elipse y la cadena de texto necesitarán dos componentes, el ancho y el largo en el caso de la elipse, y la altura y espaciado de los caracteres en el caso de la cadena de texto
`float64[] angle`	Vector de ángulos característicos. Al igual que para el vector de dimensiones, se tiene un vector que estará formado por una o dos componentes como máximo para los valores de los ángulos necesarios para definir el elemento de proyección. En el caso de la línea se necesita solamente una componente para el ángulo de la pendiente, así como para la

	elipse y la cadena de caracteres la cual representará el ángulo de rotación de la figura. Por su parte, el arco necesitará dos componentes que indicarán el ángulo de inicio y fin
`string text`	Cadena de caracteres. Solo es válido para el elemento de proyección de tipo cadena de caracteres

- CoordinateSystem.srv

Es un tipo de servicio utilizado para definir un nuevo sistema de coordenadas.

`string name`	Nombre del sistema de coordenadas
`float64 distance`	Distancia entre el proyector y la superficie de proyección
`geometry_msgs/Point[] P`	Vector de coordenadas (x,y,z) de los puntos de referencia P0, P1, P2, P3 del nuevo sistema de coordenadas, respecto al sistema de referencia del proyector {P} (coordenadas en el sistema real en mm)
`geometry_msgs/Point T0`	Coordenadas (x,y,z) del punto de referencia T0 del nuevo sistema de coordenadas, respecto al sistema de coordenadas del usuario {T}
`float64 resolution`	Resolución del nuevo sistema de coordenadas. La resolución se considera para la dimensión mayor del sistema y se calcula proporcionalmente para el resto
`---`	
`geometry_msgs/Point[] T`	Vector de coordenadas (x,y,z) de los puntos de referencia del nuevo sistema de coordenadas respecto al sistema de coordenadas del usuario {T}
`bool success`	Verdadero si el servicio ha acabado con éxito, falso en caso contrario
`string message`	Mensaje de respuesta del servicio

- CoordinateSystemName.srv

Tipo de servicio en el que se indica el nombre del sistema de coordenadas.

`string name`	Nombre del sistema de coordenadas
`---`	
`bool success`	Verdadero si el servicio ha acabado con éxito, falso en caso contrario
`string message`	Mensaje de respuesta del servicio

- CoordinateSystemList.srv

Este tipo de servicio devuelve la lista de sistemas de coordenadas definidos y el sistema de coordenadas activo en el momento de la llamada al servicio.

`------`	
`bool success`	Verdadero si el servicio ha acabado con éxito, falso en caso contrario
`string message`	Mensaje de respuesta del servicio
`string[] cs_list`	Vector de nombres de sistemas de coordenadas definidos en el proyector
`string active_cs`	Nombre del sistema de coordenadas actualmente activo

- CoordinateSystemShow.srv

En este servicio se indica el número de segundos que se quiere mostrar los ejes de coordenadas y el marco del sistema de coordenadas activo.

`int16 secs`	Número de segundos de proyección del sistema de coordenadas
`-------`	
`bool success`	Verdadero si el servicio ha acabado con éxito, falso en caso contrario
`string message`	Mensaje de respuesta del servicio

- ProjectionElement.srv

Este último tipo de servicio se utiliza para identificar un elemento de proyección a través de su tipo, el grupo al que pertenece y su nombre.

`int32 figure_type`	Identificador del tipo de figura. Vale 1 para la línea, 2 para el círculo, 3 para el arco, 4 para la elipse y 5 para la cadena de caracteres
`string projection_group`	Nombre del grupo de proyección al que pertenece el elemento
`string figure_name`	Nombre del elemento de proyección
`-------`	
`bool success`	Verdadero si el servicio ha acabado con éxito, falso en caso contrario
`string message`	Mensaje de respuesta del servicio

5.2 z_laser_zlp1

En este subpaquete se definen las librerías en las que se implementan las funcionalidades del proyector de manera que puedan ser utilizadas desde la API de una aplicación basada en ROS. De esta forma, las funciones serán accesibles desde los topics y los servicios definidos en esta API.

En cuanto a las librerías, se han desarrollado repartidas en clases, cada una dedicada a resolver un conjunto específico de funcionalidades:

- zlp_connections: se ocupa de las funciones de conexión y desconexión con el dispositivo.

- zlp_coordinate_system: engloba las funciones relacionadas con la definición y gestión de sistemas de referencia.
- zlp_projection_element: recoge las funciones relacionadas con la definición y gestión de elementos de proyección.
- zlp_keyboard: se encarga del monitoreo del teclado para la aplicación de transformaciones (traslación, rotación, escalado) a un elemento de proyección en tiempo real (al mismo tiempo que se proyecta).
- zlp_utils: ofrece algunas funciones simples útiles.
- zlp_projector_manager: gestiona las clases anteriores.

Por último, se encuentra la librería zlp_projector_ros en la que se encuentra el nodo que implementa la ROS API, es decir, donde se definen los topics y los servicios que hacen uso de las funciones presentes en las librerías anteriores. Esta API se compone de:

- Topics

~~add_projection_element (tipo 'z_laser_projector/Figure'): define un nuevo elemento de proyección asociado al sistema de referencia activo. Aun así, este no será visible hasta que se inicie la proyección.

- Servicios

~connect (tipo 'std_srvs/Trigger'): abre la conexión con el proyector.

~disconnect (tipo 'std_srvs/Trigger'): cierra la conexión con el proyector.

~projection_start (tipo 'std_srvs/Trigger'): inicia la proyección de los elementos asociados al sistema de referencia activo.

~projection_stop (tipo 'std_srvs/Trigger'): finaliza la proyección de todos los elementos.

~define_coordinate_system (tipo 'z_laser_projector/CoordinateSystem'): define un nuevo sistema de referencia a partir de los puntos de referencia indicados por el usuario.

~search_targets (tipo 'z_laser_projector/CoordinateSystem'): define un nuevo sistema de referencia mediante el escaneo de reflectores. El usuario debe proporcionar una posición aproximada de estos.

~coordinate_system_list (tipo 'z_laser_projector/CoordinateSystemList'): obtiene la lista de sistemas de referencia definidos en el proyector.

~set_coordinate_system (tipo 'z_laser_projector/CoordinateSystemName'): establece el sistema de referencia activo. El resto de sistemas definidos permanecen en segundo plano.

~remove_coordinate_system (tipo 'z_laser_projector/CoordinateSystemName'): elimina un sistema de referencia. Si se trata del sistema activo, se debe establecer uno nuevo.

~show_active_coordinate_system (tipo 'z_laser_projector/CoordinateSystemShow'): proyecta los puntos de referencia, los ejes de coordenadas y el marco del sistema de referencia activo durante un número de segundos.

~hide_projection_element (tipo 'z_laser_projector/ProjectionElement'): oculta un elemento de proyección asociado al sistema de referencia activo.

~unhide_projection_element (tipo 'z_laser_projector/ProjectionElement'): descubre un elemento de proyección asociado al sistema de referencia activo que estaba oculto.

~remove_projection_element (tipo 'z_laser_projector/ProjectionElement'): elimina un elemento de proyección asociado al sistema de referencia activo.

~monitor_projection_element (tipo 'z_laser_projector/ProjectionElement'): monitorea las transformaciones (traslación, rotación y escaldo) de un elemento de proyección mientras es proyectado en tiempo real.

- Parámetros

~projector_IP (tipo ‘string’): dirección IP del proyector.

~server_IP (tipo ‘string’): dirección IP del ZLP-Server.

~connection_port (tipo ‘int’): número del puerto de conexión.

~license_file (tipo ‘string’): ruta del archivo de la licencia.

~coordinate_system_name (tipo ‘string’): nombre del sistema de referencia activo.

~coordinate_system_resolution (tipo ‘float’): resolución respecto del sistema de referencia del usuario {T}.

~coordinate_system_distance (tipo ‘float’): distancia entre el proyector y la superficie de proyección.

~P0/x, ~P0/y, ~P1/x, ~P1/y, ~P2/x, ~P2/y, ~P3/x, ~P3/y (tipo ‘float’): posición real (mm) respecto del eje x e y del sistema de referencia del proyector {P}, de los puntos de referencia P0, P1, P2, P3.

~T0/x, ~T0/y (tipo ‘float’): posición del punto T0, origen del sistema de referencia del usuario {T}.

~projector_connected (tipo ‘bool’): estado de conexión del proyector.

~using_visualizer (tipo ‘bool’): indica si se utiliza el visualizador.

Para terminar, en la siguiente imagen se muestra un ejemplo de las proyecciones de un sistema de coordenadas y los diferentes tipos de elementos de proyección definidos por el usuario.

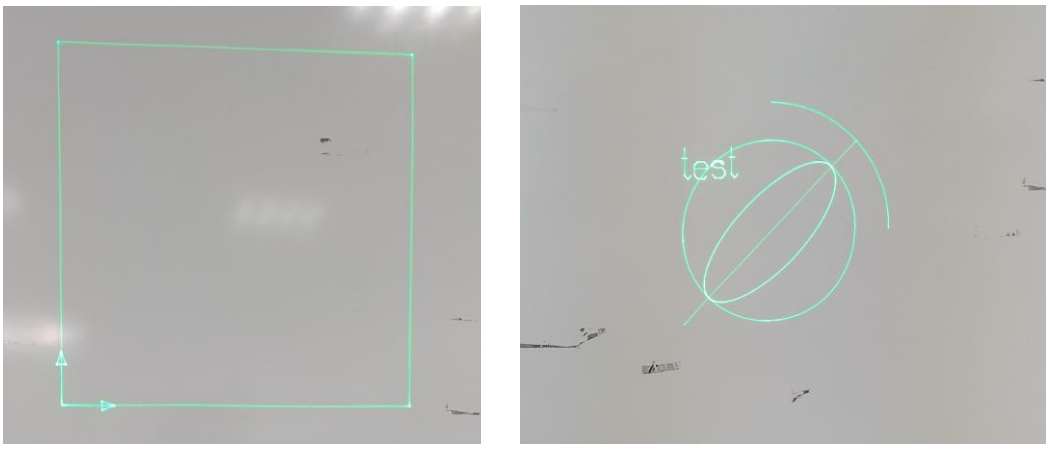


Figura 5-2. Proyección de un sistema de referencia (a la izquierda) y los diferentes tipos de elementos de proyección (a la derecha).

5.3 z_laser_viz

Este subpaquete se ocupa de ejecutar los nodos encargados de representar en ‘RViz[15]’ la posición de los sistemas de referencia (ROS-tf) y simular la proyección de las figuras creadas por el usuario (visualization markers), de manera que todos los elementos definidos en el proyector se puedan examinar constantemente desde el visualizador.

[15] http://wiki.ros.org/rviz

5.3.1 TF frames

Con el fin de representar los sistemas en el visualizador se hace uso del paquete de ROS, 'tf[16]', para crear un nodo que publica las posiciones y orientaciones de los diferentes sistemas de referencia ('frames') a lo largo del tiempo.

En este simulador se van a representar hasta un total de 4 sistemas de referencia a la vez:

- world frame: es el sistema de referencia global.
- zlp1_link frame: es el sistema de referencia asociado al modelo 3D del proyector láser y está definido respecto del sistema global. Se trata de un sistema fijo/estático ('static frame') lo que quiere decir que la posición y la rotación está predefinida y será invariable.
- [P] frame: es el sistema de referencia del proyector y está definido respecto al sistema de referencia anterior (zlp1_link frame). En este caso no es estático pero su posición se mantiene constante ya que representa el centro real del láser del dispositivo. Este sistema es útil para comparar la posición de los sistemas de referencia definidos por el usuario respecto al sistema de referencia del proyector.
- frame del sistema de referencia del usuario: tanto el nombre como la posición de este frame son variables según sea definida por el usuario y solo se representará en el visualizador el sistema de referencia activo, si es que hay alguno (el resto de sistemas definidos quedan en segundo plano).

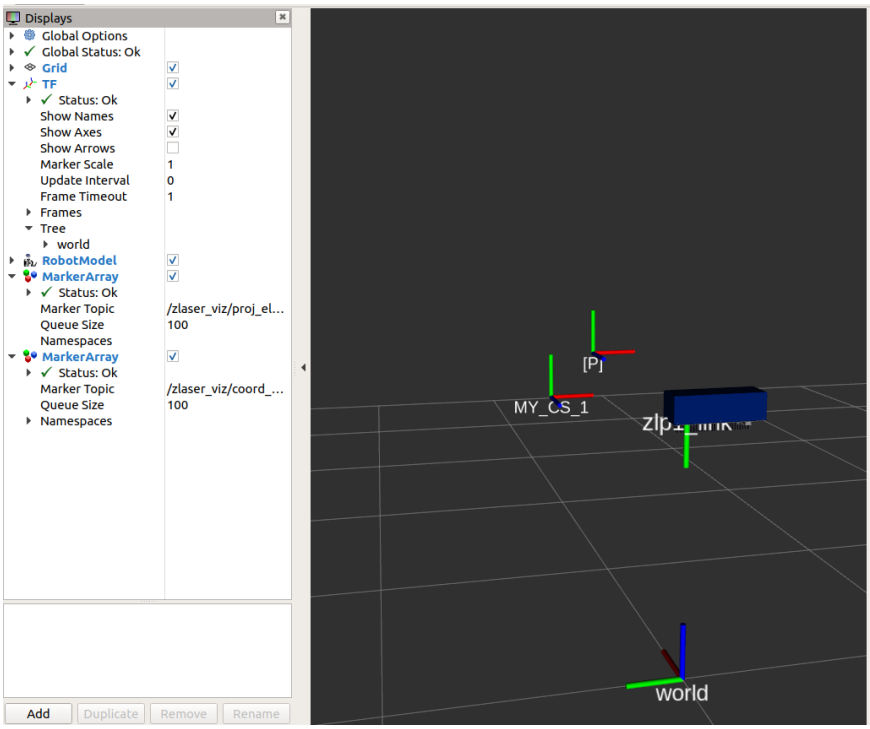


Figura 5-3. Representación de los sistemas de referencia ('frames') definidos en el visualizador y publicados mediante el paquete de ROS-tf.

5.3.2 Visualization Markers

Por otro lado, se encuentra la simulación/visualización de la proyección de los diferentes tipos de elementos que se pueden definir en el proyector por parte el usuario. Como ya se ha visto en apartados anteriores estos elementos pueden ser: líneas, círculos, arcos, elipses y cadenas de texto, aunque también se incluye en el visualizador la simulación de la proyección de los ejes de coordenadas y el marco de los sistemas de referencia.

[16] http://wiki.ros.org/tf

Para ello se utilizan los "visualization_msgs/MarkerArray[17]", un tipo de mensaje estándar definido dentro de la biblioteca "visualization_msgs" del entorno de ROS. Este tipo de mensaje está formado por un vector de "visualization_msgs/Marker" que es, a su vez, otro tipo de mensaje estándar definido en esta biblioteca.

De esta manera, lo que se tiene es un nodo que se encarga de publicar constantemente los Markers contenidos en estos vectores. Cada uno de estos Markers corresponde a un elemento de proyección creado asociado a un sistema de referencia, de tal forma que, en el momento que lo solicite el usuario, se mostrarán en el visualizador solo aquellos que les corresponda, es decir, los que estén asociados al sistema de referencia con el que se esté trabajando y que además no estén ocultos.

Para manipular las propiedades de los Markers se utilizan los parámetros definidos para este tipo de mensaje (Marker.msg), expuestos a continuación:

```
uint8 ARROW=0
uint8 CUBE=1
uint8 SPHERE=2
uint8 CYLINDER=3
uint8 LINE_STRIP=4
uint8 LINE_LIST=5
uint8 CUBE_LIST=6
uint8 SPHERE_LIST=7
uint8 POINTS=8
uint8 TEXT_VIEW_FACING=9
uint8 MESH_RESOURCE=10
uint8 TRIANGLE_LIST=11

uint8 ADD=0
uint8 MODIFY=0
uint8 DELETE=2
uint8 DELETEALL=3

Header header  # header for time/frame information
string ns  # Namespace to place this object in... used in conjunction with id to create a unique name for the object
int32 id  # object ID useful in conjunction with the namespace for manipulating and deleting the object later
int32 type  # Type of object
int32 action  # 0 add/modify an object, 1 (deprecated), 2 deletes an object, 3 deletes all objects
geometry_msgs/Pose pose # Pose of the object
geometry_msgs/Vector3 scale  # Scale of the object 1,1,1 means default (usually 1 meter square)
std_msgs/ColorRGBA color  # Color [0.0-1.0]
duration lifetime # How long the object should last before being automatically deleted.  0 means forever
bool frame_locked  # If this marker should be frame-locked, i.e. retransformed into its frame every timestep

#Only used if the type specified has some use for them (eg. POINTS, LINE_STRIP, ...)
geometry_msgs/Point[] points
#Only used if the type specified has some use for them (eg. POINTS, LINE_STRIP, ...)
```

[17] http://docs.ros.org/en/melodic/api/visualization_msgs/html/index-msg.html

```
#number of colors must either be 0 or equal to the number of points
#NOTE: alpha is not yet used
std_msgs/ColorRGBA[] colors

# NOTE: only used for text markers
string text

# NOTE: only used for MESH_RESOURCE markers
string mesh_resource
bool mesh_use_embedded_materials
```

En primera aproximación, se puede comprobar que no existe un tipo de Marker disponible para cada tipo básico de elemento de proyección admitido en el dispositivo, sin embargo, sí que se puede hacer uso de uno de ellos para representar prácticamente cualquier tipo de figura, es el caso del "LINE_STRIP". Este tipo de Marker une, mediante líneas rectas, los puntos de una trayectoria, de tal forma que si los puntos están lo suficientemente juntos se representará como una trayectoria continua y suave, obteniéndose así figuras como curvas, círculos, elipses, rectas, polígonos, etc.

Haciendo uso de este tipo de Marker, se calcula la trayectoria de puntos mediante las ecuaciones paramétricas para crear cada figura geométrica:

$$Línea\ recta: \begin{cases} x = x_0 + \lambda * \cos(\alpha_0) \\ y = y_0 + \lambda * \sin(\alpha_0) \end{cases} \quad (5\text{–}1)$$

Donde (x_0, y_0) es el punto inicial de la recta, α_0 es el ángulo de la pendiente y λ es el parámetro de iteración que va desde 0 hasta el valor de la longitud de la recta

$$Círculo\ y\ arco: \begin{cases} x = x_0 + r * \sin(\lambda) \\ y = y_0 + r * \cos(\lambda) \end{cases} \quad (5\text{–}2)$$

Donde (x_0, y_0) es centro del círculo o el arco, r es el radio y λ es el parámetro de iteración que en el caso del círculo va desde 0 hasta 360º (2π radianes) y en el caso del arco va desde el ángulo inicial al ángulo final

$$Elipse: \begin{cases} x = x_0 + a * \cos(\lambda) * \cos(\alpha_0) - b * \sin(\lambda) * \sin(\alpha_0) \\ y = y_0 + a * \cos(\lambda) * \sin(\alpha_0) + b * \sin(\lambda) * \cos(\alpha_0) \end{cases} \quad (5\text{–}3)$$

Donde (x_0, y_0) es centro de la elipse, a es el ancho, b es el largo, α_0 es el ángulo de rotación y λ es el parámetro de iteración que va desde 0 hasta 360º

En el caso del elemento de proyección de cadena de texto, sí que se utiliza el tipo TEXT_VIEW_FACING directamente.

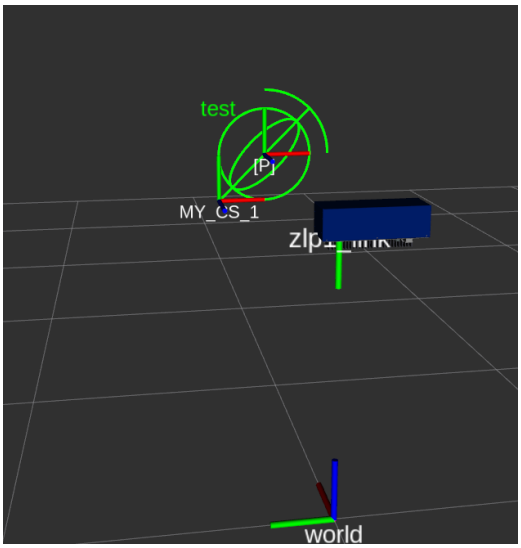


Figura 5-4. Proyección de los sistemas de referencia y los diferentes tipos de elementos de proyección en el visualizador.

En la siguiente imagen se muestra un ejemplo de aplicación del visualizador en el que se puede comprobar la proyección del contorno de un hueco para una pieza del fancowl de un avión.

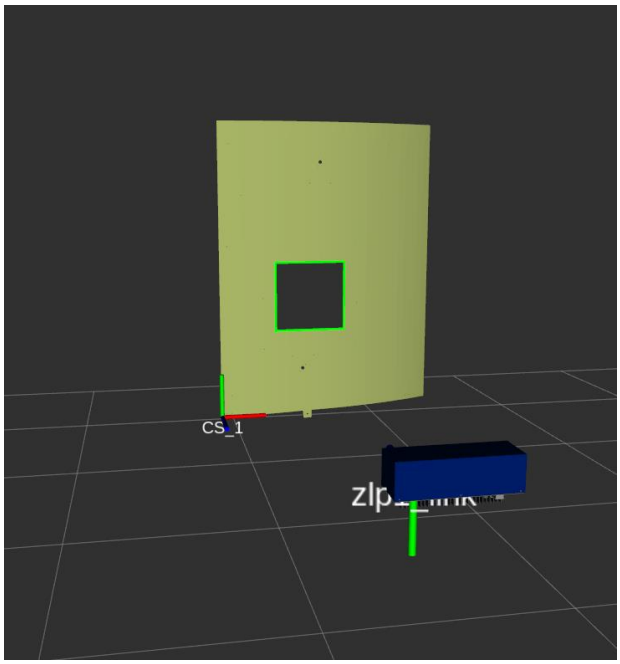


Figura 5-5. Ejemplo de uso del visualizador en el que se proyecta el contorno del hueco en una pieza aeronáutica.

5.4 z_laser_gui

Como casi en cualquier otra aplicación o paquete, la API de ROS, o lo que es lo mismo, los topics y los servicios son accesibles desde el terminal. Sin embargo, para usuarios que están poco familiarizados con este entorno puede no resultar del todo intuitivo.

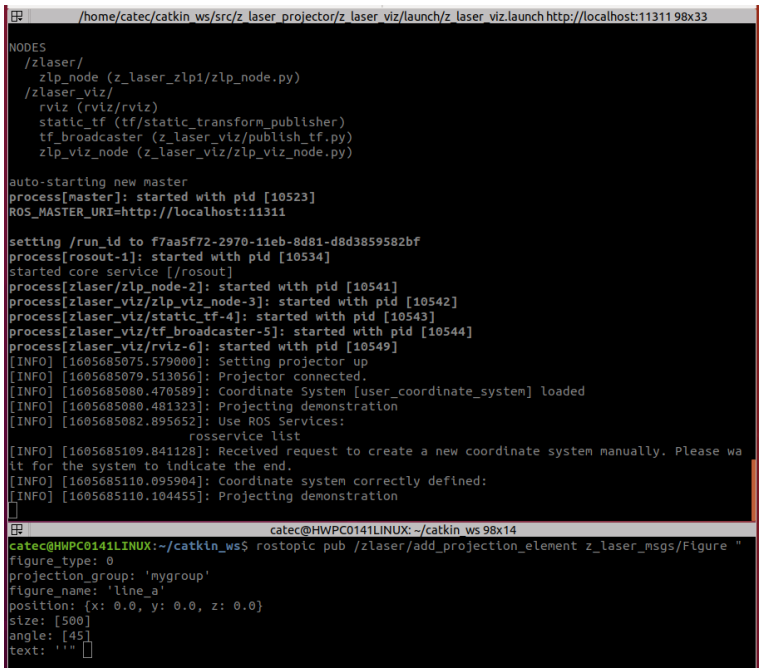


Figura 5-6. Interacción con la ROS API del paquete desde el terminal.

En este punto, para atraer la atención de más fabricantes e industrias, así como para facilitar la tarea de los operarios de interactuar con el paquete y ejecutar las funcionalidades que este ofrece, se ha desarrollado una interfaz, utilizando la herramienta QtDesigner[18], que en términos generales lo que hace es sintetizar el envío de mensajes y la llamada a los servicios mencionados en apartados anteriores en una interfaz gráfica más sencilla y amigable, aportando un nivel de abstracción mayor que evita la necesidad de utilizar el terminal.

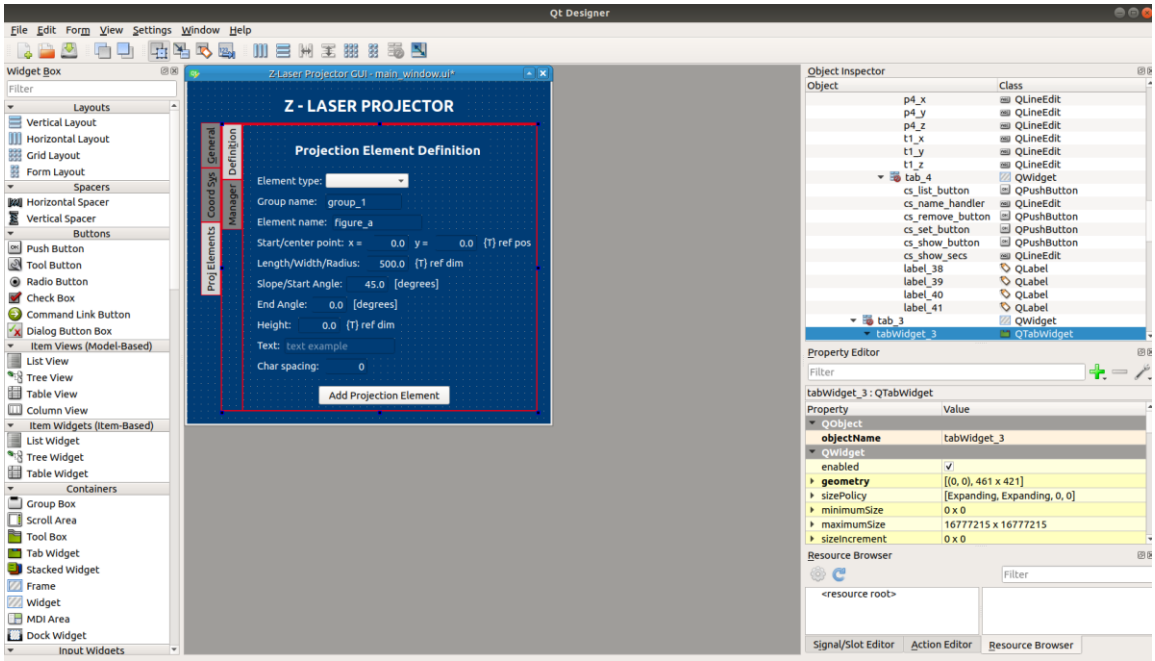


Figura 5-7. Vista de las opciones de objetos de la herramienta QtDesigner.

Como se puede ver en la imagen, esta herramienta lo que permite es diseñar gráficamente una interfaz con diferentes tipos de objetos: botones, cuadros de texto de entrada y salida, listas desplegables, navegación entre menús, etc. Una vez diseñada, se guarda en un archivo de tipo predeterminado ".ui" (formato XML) que se puede convertir después en una librería/módulo de Python.

Este módulo autogenerado lo que contiene es una clase cuyos atributos son todos los objetos insertados en la interfaz, de manera que este, junto con la librería PyQt5, permiten hacer uso de los diferentes objetos, cada cual proporciona una serie de propiedades y funcionalidades que pueden ser instanciadas en tiempo de ejecución. Para autogenerar este módulo desde el tipo predeterminado (".ui"), se debe ejecutar el siguiente comando:

```
$ pyuic5 main_window.ui -o main_window.py
```

Para este paquete concretamente la interfaz diseñada se compone de 3 menús principales:

- General:

En este menú se recogen las funciones más generales que ofrece el proyector como son: la conexión (~connect service) y desconexión (~disconnect service) del dispositivo, y el inicio (~projection_start service) y paro (~projection_stop service) de la proyección de los elementos asociados al sistema de coordenadas activo.

[18] https://doc.qt.io/qt-5/qtdesigner-manual.html

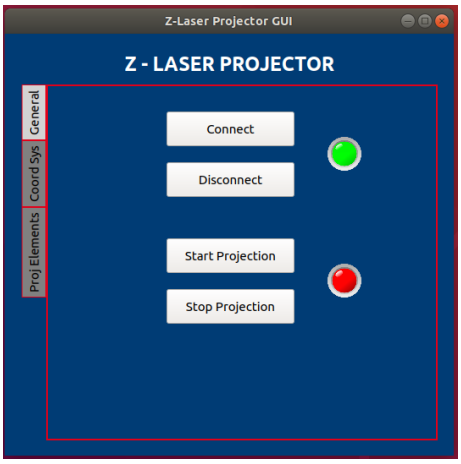


Figura 5-8. Menú general de la GUI.

- Sistemas de Coordenadas:

En este segundo menú se recoge por un lado la definición de un sistema de coordenadas nuevo, ya sea definiendo los puntos de referencia de forma manual (~define_coordinate_system service) o mediante el escaneo automático de reflectores (~search_targets service). Por otro lado, se tiene un segundo submenú en el que se encuentran las funcionalidades relacionadas con la gestión de los sistemas de coordenadas definidos como son: establecer el sistema de coordenadas activo (~set_coordinate_system service), mostrar los ejes de coordenadas y el marco de este sistema (~show_coordinate_system service), eliminar un sistema de coordenadas (~remove_coordinate_system service) y obtener la lista de los sistemas definidos en el proyector (~coordinate_system_list service).

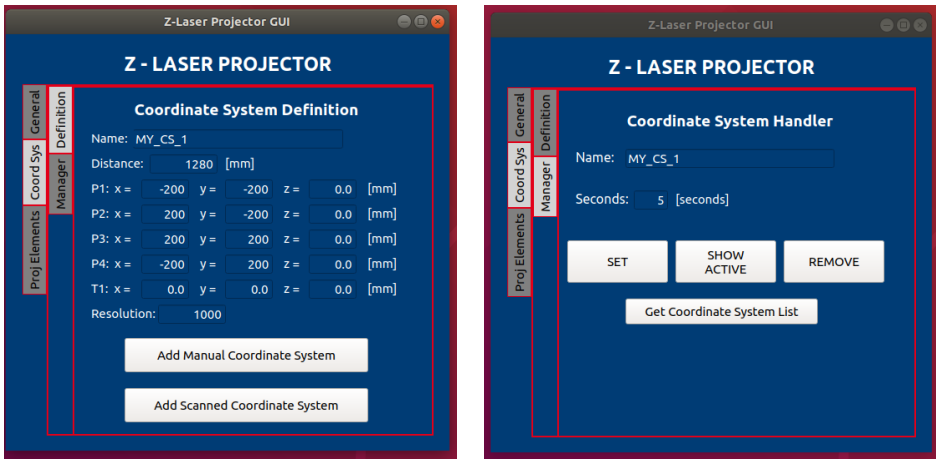


Figura 5-9. Submenús dedicados a los sistemas de coordenadas.

- Elementos de Proyección:

Este último menú, como el anterior, tiene un primer submenú en el que se recoge la definición de los diferentes tipos de elementos de proyección básicos: línea, círculo, arco, elipse y cadena de caracteres (~add_projection_element topic). Para cada uno de ellos se van mostrando los campos que es necesario rellenar para su correcta definición. De la misma manera, se tiene un segundo submenú en el que aparecen las operaciones de gestión de elementos de proyección: activar (~hide_projection_element service), desactivar (~unhide_projection_element service), eliminar (~remove_projection_element service) y monitorear por teclado un elemento de proyección (~monitor_projection_element service).

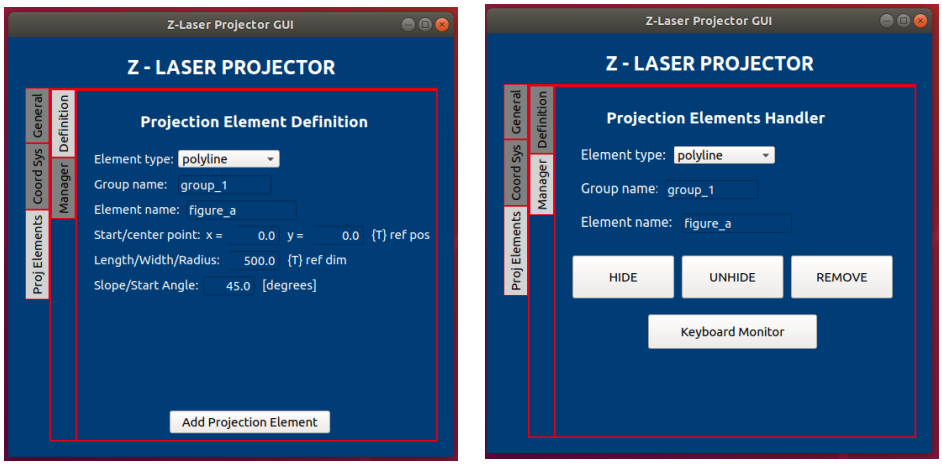


Figura 5-10. Submenús dedicados a los elementos de proyección.

Para terminar, la GUI, también gestiona y muestra los mensajes de error generados durante la ejecución, en forma de ventanas emergentes.

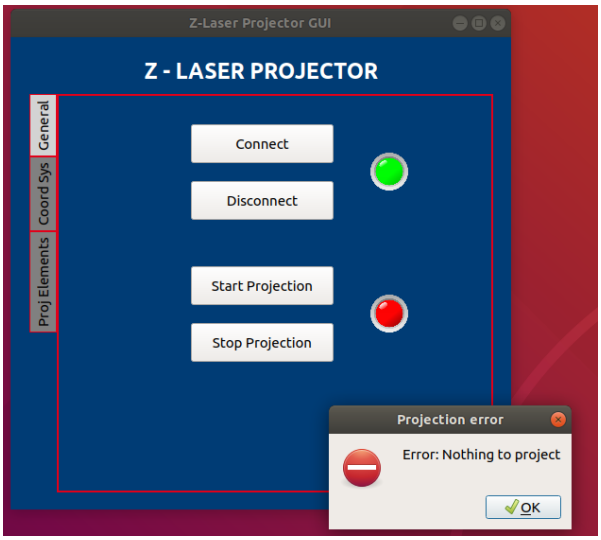


Figura 5-11. Ventana emergente con un mensaje de error producido al iniciar la proyección sin haber definido ninguna figura.

6 GARANTÍA DE CALIDAD

Las comunidades y el paradigma del software de código abierto ('Open Source Software', OSS) está cobrando fuerza y es adoptado cada vez más por la industria del software tradicional, convirtiéndolo en un serio contendiente para el suministro de software comercial[22]. Este interés industrial trae consigo sus propios requisitos para los OSS[23], especialmente en lo que respecta a la calidad.

Cuando se habla de un software de calidad, en el contexto de la ingeniería de software, se refiere a un software que aporte valor, es decir, que no tiene defectos, es confiable y se integra rápidamente [X].

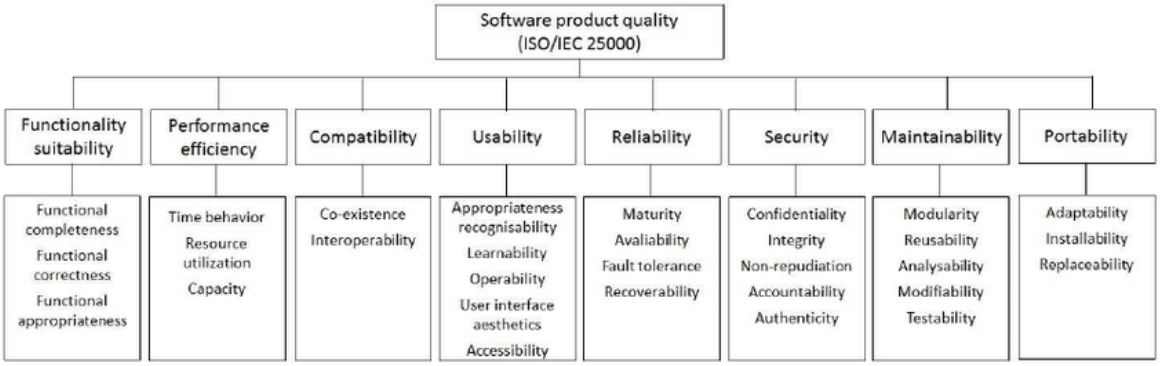


Figura 6-1. Propiedades de calidad del software por la ISO/IEC 25010:2011[24].

Para lograrlo, las organizaciones tradicionales utilizan una combinación de prácticas, procesos y técnicas que permiten garantizar esta calidad del software (QA, 'Quality Assurance'). Sin embargo, no está tan clara la forma en que las comunidades de OSS perciben la calidad ni los problemas que plantea la aplicación de estas prácticas en el ámbito de estas comunidades, ya que este tipo de organizaciones comunitarias son culturalmente diferentes y se han establecido sobre la base de un conjunto de valores y objetivos fundamentalmente distintos[25].

En el estudio realizado por Halloran y Scherlis[26] se evaluaron once proyectos de OSS para determinar las prácticas de garantía de calidad adoptadas por las comunidades. En este estudio se observó una variación en la adopción y aplicación de estas: mientras que algunas comunidades como, por ejemplo, Mozilla o NetBeans, tienen equipos dedicados a la garantía de la calidad y procedimiento bien establecidos, otras comunidades parecen seguirlas sólo de forma rudimentaria. Lamentablemente, no se puede extraer un patrón claro sobre cuándo una práctica tiene éxito y cuándo fracasa. En otro estudio realizado por Michlmayr[27], en el que se estudiaron las prácticas de calidad en siete comunidades de OSS, se concluyó que el grado de adopción de estas influye en la calidad del producto final.

Entonces, ¿cómo se puede asegurar la calidad en el desarrollo de OSS? La garantía de calidad se basa en: un conjunto bien establecido de prácticas, definiendo el término práctica como una forma común de actuar, reconocida por la comunidad como la forma correcta de hacer las cosas; en el rigor de los procedimientos; y en la exhaustividad y alta cobertura de las pruebas. Además, la calidad depende en gran medida de la alta participación en el proyecto, ya que esto facilita el descubrimiento de errores generando un ciclo de retroalimentación en el que los defectos se identifican y corrigen más rápidamente.

Como ya se ha comentado, la comunidad de ROS es un ejemplo de comunidad OSS grande y diversa: su plataforma Wiki recibe más de 1,4 millones de visitantes únicos al año y tiene 6.749 usuarios registrados, el foro de la comunidad recibe un promedio de 150 mensajes a la semana, y el total de descargas de los paquetes Debian es de más de 13,4 millones.

Esta comunidad, a pesar de ser una comunidad multidisciplinaria, donde la mayoría de los desarrolladores de ROS no son ingenieros de software, sino que sus miembros tienen formación y experiencias profesionales diversas, desde desarrolladores de paquetes, hasta estudiantes, investigadores, fabricantes, principalmente de disciplinas de mecánica, robótica y electrónica, posee una fuerte conciencia de calidad[28].

Por esta razón, surge ROSIN, una iniciativa cuyo objetivo, como ya se comentó, es el de crear herramientas que permitan acelerar el uso e integración de ROS, incluyendo el desarrollo de nuevas utilidades y la mejora de las prácticas, ya existentes, postuladas para garantizar la calidad de las aplicaciones robóticas aportadas por la comunidad de ROS.

Entre estas utilidades y prácticas[19] a las que hace referencia la iniciativa ROSIN, las cuales algunas de ellas se desarrollarán más adelante, se encuentran:

- Hacer visible la calidad de los paquetes ROS mediante una herramienta donde la calidad de los paquetes pueda ser medida, asignada y mostrada.
- Dinamizar el proceso de revisión del código.
- Implementar un método y una herramienta de exploración de código.
- Cuestiones de mantenimiento: atraer y reclutar nuevos mantenedores y reducir el número de paquetes huérfanos.
- Energizar la Integración Continua (CI), revisando y mejorando la actual implementación de estos servicios.
- Sitio web para el Quality Hub: será el lugar central de "acceso" para el intercambio de conocimientos y documentación de las prácticas de QA.
- Formalizar el proceso de propiedad del código.
- Proceso de incorporación para los miembros de la comunidad básica y no básica.
- Pruebas de modelo en bucle (‘model-in-the-loop’): esto identificará y demostrará las oportunidades de utilizar el paradigma de desarrollo basado en modelos para la generación de código.
- Implementar un proceso de mejora continua en un esfuerzo continuo por mejorar las prácticas de garantía de calidad. Estos esfuerzos son la revisión y mejora incremental de las prácticas ya existentes.
- Generación de pruebas (tests) unitarias automatizadas: el objetivo de esta iniciativa es automatizar la creación de estos tests.
- Un foro dedicado a la garantía de calidad.
- Entre otras.

A continuación, de las prácticas postuladas para asegurar la calidad del software, se desarrollan con más detalle las más se emplean en la comunidad[20][21] y además han sido utilizadas a lo largo de este proyecto:

6.1 Control de versiones

Una de las prácticas más comunes y básicas es el control de código fuente o también denominado control de versiones. Se trata de una práctica basada en el seguimiento y administración de los cambios en el código fuente. Al mismo tiempo, se habla de los sistemas de control de versiones (VCS) con los que se refiere al software que proporciona las herramientas que ayudan a poner en práctica esta técnica.

Estos sistemas VCS, como por ejemplo Git, svn, hg, cvs, bzr, etc., mantienen un historial del desarrollo del código de manera que es posible supervisar los cambios, consultar el historial de revisiones, volver a versiones anteriores, resolver conflictos al combinar las contribuciones de diferentes orígenes, entre otras funcionalidades. De esta forma, los sistemas de control de versiones ayudan a optimizar el proceso de

[19] https://www.rosin-project.eu/software-quality-assurance
[20] http://wiki.ros.org/Quality/Tutorials
[21] http://wiki.ros.org/DevelopersGuide

desarrollo y proporcionan un origen centralizado, facilitando el trabajo en equipo y asegurando que los desarrolladores siempre estén trabajando en la versión correcta del código.

Aunque ROS no obliga a los desarrolladores a utilizar ningún sistema de control de versiones específico ni ningún rastreador de errores, a partir de 2012 se establecieron unas pautas recomendadas para esta práctica, basadas en Git, por diversas razones:

- Los usuarios/desarrolladores solo necesitan conocer una única herramienta y plataforma para buscar la mayoría de las aplicaciones desarrolladas en ROS.
- Administrar parches ('patch'), fusiones ('merge'), peticiones de validación ('pull request'), etc. desde el mismo lugar.
- Administrar también el seguimiento de problemas ('issues') o errores ('bugs') desde ese mismo lugar.
- Fomentar la comunicación de los desarrolladores en toda la comunidad ROS con el fin de homogeneizar la base de código y que sea más fácil navegar.

Entre estas pautas se puede encontrar:

- No añadir archivos generados por el ordenador, como objetos (.o), bibliotecas (.a, .so, .dll), o scripts de configuración autogenerados.
- No añadir archivos binarios grandes: subirlos a un servidor web y descargarlos.
- Confirmar los cambios ("commit") en el código a menudo.
- Intentar mantener las confirmaciones centradas en un cambio en particular en lugar de agrupar varios cambios juntos.
- Verificar que el código compila antes de confirmar los cambios.
- etc.

Los motivos por lo que ROS se ha estandarizado en la tecnología de Git[22] como el VCS preferido frente a sus predecesores son:

- Múltiples herramientas VCS: una gran cantidad de sistemas de control de versiones dificultaban el intercambio de código.
- Git es un sistema de control de versiones distribuido (DVCS) de código abierto. Este tipo de VCS tiende a funcionar mejor en el desarrollo de código distribuido a gran escala en contraste con un sistema de control de versiones centralizado (CVCS).
- Git tiene aproximadamente la misma (si no más) funcionalidades que sus rivales.
- Pobre velocidad y tiempo de actividad: ROS ha pasado por muchos servicios de alojamiento VCS, tanto externos (SourceForge) como internos (code.ros.org, kforge.ros.org) y todos han presentado inconvenientes. Sin embargo, la plataforma GitHub[23], un servicio de alojamiento para proyectos estrechamente vinculados con Git, se ha convertido rápidamente en el servicio de alojamiento dominante para proyectos de código abierto por la alta velocidad que ofrece. De hecho, es ahora el sitio de alojamiento oficial de los paquetes principales de ROS.
- Gastos generales de mantenimiento: code.ros.org y kforge.ros.org fueron mantenidos por el equipo principal de ROS, generando una sobrecarga de mantenimiento significativa.
- Es difícil trabajar con la comunidad: fue difícil incorporar comentarios y parches de la comunidad debido a la naturaleza heterogénea de las herramientas de VCS, los servicios de alojamiento y los rastreadores de errores.
- Rastreadores de errores dispersos: anteriormente se gestionaban múltiples instancias de rastreadores de errores (por ejemplo, trac, bugzilla) que no eran uniformes en todos los paquetes.

[22] https://git-scm.com/
[23] https://github.com/

Con estas indicaciones, en el desarrollo del paquete de este trabajo también se ha hecho uso de Git como herramienta para el control de versiones del código, así como de la plataforma GitHub para alojar el repositorio de forma pública que se puede encontrar en la siguiente dirección web: https://github.com/fada-catec/z_laser_projector.

6.1.1 Git Branching Workflow

Un flujo de trabajo ('workflow') de Git es una fórmula o una recomendación acerca del uso de esta tecnología para realizar el trabajo de forma uniforme y productiva. Git ofrece a los usuarios una amplia flexibilidad de gestión de cambios y estos flujos de trabajo animan a los usuarios a sacar partido a la herramienta de forma eficaz y estable. Como resultado de esta flexibilidad, no existe un proceso estandarizado acerca de cómo interactuar con esta tecnología, pero cuando se trabaja con un equipo en un proyecto gestionado con Git, es importante asegurarse de que todo el equipo está de acuerdo en cómo se va a aplicar el flujo de cambios para garantizar que todo el equipo se encuentra en sintonía, de modo que lo primero que se hace es desarrollar o seleccionar un tipo de flujo de trabajo.

Existen varios ejemplos publicados de flujos de trabajo para Git, entre los que se encuentra el flujo de trabajo basado en ramas de funciones[24]. La idea principal que subyace a este tipo de flujo de trabajo es que el desarrollo de una funcionalidad debe llevarse a cabo en una rama especializada, en lugar de en la rama principal. Este aislamiento permite que varios desarrolladores trabajen en una función concreta sin perturbar el contenido del código base principal. También implica que la rama principal no debe contener en ningún caso código erróneo, lo que supone una gran ventaja para los entornos de integración continua.

En el paquete Z_LASER_PROJECTOR se utiliza este modelo de flujo de trabajo donde se ha creado una rama principal para cada versión de ROS desarrollada del paquete (melodic y kinetic) y sobre las que se han ido desarrollando las funcionalidades correspondientes en ramas derivadas.

6.2 Guías de estilo

Los nombres juegan un papel importante en ROS y seguir las convenciones[25] de nomenclatura simplifica el proceso de aprendizaje y comprensión de sistemas grandes.

En este apartado se mencionan algunas de las convenciones para los recursos comunes de ROS:

- Nomenclatura de paquetes: Los paquetes ROS ocupan un espacio de nombres plano, por lo que la asignación de nombres se debe realizar de forma cuidadosa y coherente.
    - Los nombres de los paquetes deben seguir las convenciones comunes de nomenclatura de variables en C: minúsculas, comenzar con una letra, usar guion bajo para separar. Por ejemplo: laser_viewer.
    - Los nombres de los paquetes deben ser lo suficientemente específicos para identificar lo que hace el paquete.
    - El nombre de un paquete no debe contener "ROS" ya que es redundante.
    - No usar un nombre que ya haya sido utilizado.
- Nomenclatura de topics y servicios: Los nombres de los topics y los servicios se encuentran en un espacio de nombres jerárquico y las bibliotecas de cliente proporcionan mecanismos para reasignarlos en tiempo de ejecución, por lo que hay más flexibilidad que con los paquetes. Sin embargo, es mejor minimizar la necesidad de espacios de nombres y reasignación de nombres.
    - Los nombres de topics y servicios deben seguir las convenciones comunes de declaración de variables en C: minúsculas, separación con guion bajo. Por ejemplo, laser_scan.
    - Los nombres de los topics y servicios deben ser razonablemente descriptivos.

[24] https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow
[25] http://wiki.ros.org/ROS/Patterns/Conventions#Naming_ROS_Resources

- Nomenclatura de mensajes:
    - Los archivos de mensajes se utilizan para determinar el nombre de la clase del código generado automáticamente. Como tal, deben ser CamelCased. Por ejemplo: LaserScan.msg
    - Sin embargo, los campos del mensaje deben estar en minúsculas con guiones bajos para separar. Por ejemplo: range_min.
- Nomenclatura de nodos: Los nodos tienen un tipo y un nombre. El tipo es el nombre del ejecutable para iniciar el nodo, mientras que el nombre es lo que se pasa a otros nodos cuando se inicia. Se separan estos dos conceptos porque los nombres deben ser únicos, mientras que puede tener varios nodos del mismo tipo.
    - Cuando sea posible, el nombre predeterminado de un nodo debe seguir del nombre del ejecutable utilizado para iniciar el nodo. Este nombre predeterminado se puede reasignar al inicio a algo único.
    - En general, se recomienda que los nombres de los tipos de nodo sean cortos porque están dentro del ámbito del nombre del paquete. Por ejemplo, si el paquete laser_scan tiene un visualizador para los escaneos del láser, se puede llamar simplemente view en lugar de laser_scan_viewer.

Por otro lado, se encuentra la guía de estilo[26] que se debe seguir al escribir código Python para ROS. El código debe seguir la guía PEP 8[27] que no es una guía de estilo estricta y valora la legibilidad sobre la coherencia:

- package_name
- ClassName
- method_name
- field_name
- _private_something
- self.__really_private_field
- _global
- 4 space indentation

Otras pautas de estilo para paquetes desarrollados con Python son:

- Todo el código Python que sea código fuente se debe encontrar bajo la misma carpeta/módulo. ROS exporta el directorio fuente (src) para que esté en la ruta de las dependencias del paquete. Se recomienda que el nombre de este módulo sea el mismo que el del paquete.

```
packagename
 |- src/
    |- packagename/
      |- __init__.py
      |- yourfiles.py
 |- scripts/
    |- non-exported python files
```

- El nombre de un tipo de nodo es el mismo que el nombre de su ejecutable. Típicamente, para los archivos Python, esto significa incluir el 'shebang': *#!/usr/bin/env Python*, en la parte superior del script principal, y que el nombre de este script sea el mismo que el nombre del nodo.

[26] http://wiki.ros.org/PyStyleGuide
[27] https://www.python.org/dev/peps/pep-0008/

6.2.1 Escáner de código

'Linting' es el proceso de ejecutar un programa que analice el código en busca de posibles errores.

Originalmente lint era el nombre de una herramienta de programación utilizada para detectar código sospechoso, confuso o incompatible entre distintas arquitecturas en programas escritos en C; es decir, errores de programación que escapan al habitual análisis sintáctico que hace el compilador. En la actualidad, el término se utiliza genéricamente para designar a herramientas que realizan estas tareas de comprobación en cualquier lenguaje de programación. Las herramientas de tipo lint generalmente funcionan realizando un análisis estático del código fuente.

Las construcciones sospechosas que suelen buscar estas herramientas son: usos de variables antes de ser inicializadas o creadas, condiciones que no varían bajo ninguna circunstancia, cálculos cuyos resultados probablemente caigan fuera del rango permitido por las variables utilizadas, entre otras.

Las sucesivas generaciones de herramientas del estilo de lint han seguido ampliando el rango de construcciones incorrectas o sospechosas y las herramientas más avanzadas realizan cada vez más comprobaciones, como, por ejemplo, que el código sea consistente entre distintos compiladores, o el soporte para incorporar anotaciones acerca del comportamiento esperado o las propiedades del código.

Para Python, que es el lenguaje de código empleado principalmente en este trabajo, existen diferentes linters que se pueden habilitar en Visual Studio Code (IDE). Pylint[28] es el linter que usa este IDE de forma predeterminada, y también ha sido el que se ha utilizado durante todo el desarrollo del paquete Z_LASER_PROJECTOR.

6.3 Automatización de tests

En este apartado se exponen los fundamentos, mejores prácticas y políticas para escribir y ejecutar tests unitarios (unit tests) y tests de integración (integration tests).

Para entender la utilidad de los tests automatizados se supone el siguiente escenario: un desarrollador prueba en su ordenador que un parche a un error en un paquete central de ROS (por ejemplo, roscpp) funciona correctamente. Si automatiza la prueba y la envía junto con el parche, facilitaría enormemente a los demás desarrolladores la tarea de evitar la reintroducir el error de nuevo. Además, el servicio de integración continua, del que se hablará más adelante, será capaz de detectar tales regresiones automáticamente.

A continuación, se exponen algunas de las razones por las que se deben implementar tests automatizados:

- Se pueden hacer actualizaciones incrementales del código más rápidamente. Si el cambio pasa los tests, se puede estar seguro de que no se ha introducido ningún problema.
- Se puede refactorizar el código con mayor confianza.
- Se mejora el diseño del código: los tests obligan a escribir el código para que sea más fácil de probar.
- Se evitan errores recurrentes (regresión de errores). Además, es mucho más fácil convencer a quien deba aceptar el parche de que el problema está resuelto, y la contribución es de alta calidad.
- Desarrollo basado en contratos: si el parche pasa las pruebas escritas por otros, se puede reclamar que se hizo bien el trabajo.
- Permite que otras personas puedan trabajar en su código más fácilmente ya que con los tests se pueden validar sus cambios. Así, los tests se convierten en una forma de documentación del código que no necesita ser leída durante la mayor parte del tiempo, y cuando necesita ser inspeccionada, el sistema de pruebas indicará con precisión qué leer.
- Simplifican el mantenimiento, especialmente para los paquetes maduros, que cambian más lentamente, y que en su mayoría necesitan ser actualizados a nuevas dependencias. Un conjunto de tests ayuda a establecer rápidamente si el paquete todavía funciona.

[28] https://pypi.org/project/pylint/

- Amplifican el valor de la integración continua. Los tests de regresión, junto con los tests de requisitos basados en escenarios, contribuyen a los tests automatizados para el paquete. Esto aumenta la efectividad del sistema de compilación y de la integración continua (CI).

Los tests automatizados son importantes para facilitar la colaboración de muchos desarrolladores en un proyecto de código abierto distribuido como es ROS. El problema es que a menudo los desarrolladores reintroducen errores porque se olvida el fundamento de una determinada decisión en el código, y los errores vuelven a aparecer (errores de regresión). Por esta razón, cuando se contribuye con código a ROS, ya sea con correcciones de errores o con nuevas características, hay que incluir tests automatizados.

Para la implementación de tests en ROS[29] se utilizan diferentes herramientas:

- Para probar el código a nivel de biblioteca se utiliza el módulo de tests unittest[30] en el caso de Python, o el módulo de tests de Google, gtest[31] para C++. Estos módulos son genéricos para el desarrollo de tests en estos lenguajes y se utilizan de forma muy extendida en múltiples campos del desarrollo de software. Para que estas herramientas sean compatibles con ROS, se utiliza, además, el módulo rosunit[32] que añade una capa adicional que permite la salida en formato XML de los resultados del test.
- Para los tests a nivel de nodo de ROS, es decir, que involucran a ROS como un middleware de comunicación, se utiliza rostest[33], junto con uno de los anteriores, unittest o gtest. Esto se aplica tanto a los tests de nodo único como a los tests que requieren la integración de varios nodos (técnicamente conocidos como tests de integración, a diferencia de los tests unitarios).

Es fundamental que los tests no sólo sean automáticos, sino que se integren en los scripts del proyecto, de modo que sean ejecutados por la infraestructura de compilación y tests, siempre que el proyecto esté siendo probado. Para ejecutar los tests, se necesita la integración catkin/roslaunch, que se verá en el siguiente apartado.

Con estas herramientas se pueden desarrollar los siguientes niveles de tests dependiendo de su alcance:

- Nivel 1 - Test unitario a nivel de código (unittest/gtest a través de rosunit). Un test unitario de código debería probar el código sin ROS. Estos tests unitarios son una forma de comprobar el correcto funcionamiento de una unidad de código. Por ejemplo, desde el punto de vista del diseño funcional se comprueba una función o un procedimiento, mientras que desde el punto de vista de diseño orientado a objetos se comprueba una clase. Esto sirve para asegurar que cada unidad funcione correcta y eficientemente por separado.
- Nivel 2 - Test unitario a nivel de nodo de ROS (rostest + unittest/gtest). Los tests unitarios de nodo ponen en marcha el nodo y prueban la API externa, es decir, topics publicados, topics suscritos y servicios.
- Nivel 3. Test de integración/regresión de nodos ROS (rostest + unittest/gtest). Los tests de integración ponen en marcha múltiples nodos y comprueban que todos ellos funcionan juntos como se espera. Depurar una colección de nodos ROS es como depurar un código multihilo: puede haber un punto muerto, etc. Una prueba de integración es a menudo la mejor manera de descubrir los errores.

Los niveles en los que el código necesita ser probado dependerá del tipo de código, pero siempre es mejor escribirlos en el nivel más bajo posible donde se exhibe el problema. Por ejemplo: una librería OSS de ROS probablemente será un test de nivel 1; un servicio de ROS sólo tendrá que ser probado en los niveles 1 y 2, mientras que un subscriptor de nodos necesitará ser probado en los tres niveles. La razón de esto es triple: en primer lugar, las pruebas de nivel inferior son más eficientes, implican menos infraestructura ROS, y por lo tanto su ejecución es más rápida, lo que es beneficioso tanto de forma local como en la integración continua. En segundo lugar, las pruebas de nivel inferior localizan mejor el problema, por lo que cuando fallan, es más fácil para los nuevos desarrolladores diagnosticar lo que está pasando. Y tercero, las pruebas más localizadas,

[29] http://wiki.ros.org/Quality/Tutorials/UnitTesting
[30] http://wiki.ros.org/unittest
[31] http://wiki.ros.org/gtest
[32] http://wiki.ros.org/rosunit
[33] http://wiki.ros.org/rostest

introducen un menor costo de mantenimiento, ya que tienen menos dependencias.

6.3.1 Tests unitarios (nivel 1 y nivel 2)

Como se ha visto en el apartado anterior se pueden encontrar dos tipos de tests unitarios, los de nivel 1 y los de nivel 2:

- Tests unitarios a nivel de código (nivel 1): estas son los tests unitarios típicos. En general, realizan llamadas directas al código donde el script de prueba no es un nodo.
- Tests unitarios a nivel de nodo de ROS (nivel 2): para probar la API externa de topics y servicios de un nodo. Para estos tests, se supone que el script de prueba es en sí mismo un nodo.

La sintaxis para ejecutar estos dos tipos de tests es diferente, por lo que es importante distinguirla correctamente. Los tests a nivel de nodo generarán recursos ROS adicionales, mientras que los tests a nivel de código serán más ligeros.

- Sintaxis para un test unitario a nivel de código:

```
#!/usr/bin/env python
PKG='test_foo'
import roslib; roslib.load_manifest(PKG)  # This line is not needed with Catkin.
import sys
import unittest

## A sample python unit test
class TestBareBones(unittest.TestCase):
    def test_one_equals_one(self):
        self.assertEquals(1, 1, "1!=1")

if __name__ == '__main__':
    import rosunit
    rosunit.unitrun(PKG, 'test_bare_bones', TestBareBones)
```

- Sintaxis para un test unitario a nivel de nodo:

```
#!/usr/bin/env python
PKG = 'test_roslaunch'
import roslib; roslib.load_manifest(PKG)  # This line is not needed with Catkin.
import sys
import unittest

## A sample python unit test
class TestBareBones(unittest.TestCase):
   ## test 1 == 1
   def test_one_equals_one(self): # only functions with 'test_'-prefix will be run!
       self.assertEquals(1, 1, "1!=1")

if __name__ == '__main__':
   import rostest
```

```
    rostest.rosrun(PKG, 'test_bare_bones', TestBareBones
```

6.3.2 Tests de integración (nivel 3)

Como se ha comentado, los tests de integración hacen uso del módulo rostest para poner en marcha múltiples nodos y comprobar que todos ellos funcionan juntos como se espera.

Para ello, rostest utiliza unos archivos que son compatibles con roslaunch y generalmente tienen una extensión ".test", aunque también puede ser ".launch". La diferencia entre rostest y roslaunch es que rostest también procesa las etiquetas "<test>", que sirven para especificar, de los nodos que se van a ejecutar, cuáles son de prueba.

La sintaxis de un archivo ".test" en el que un nodo "test_mynode" es un nodo de test se describiría como sigue:

```
<launch>
  <node pkg="mypkg" type="mynode" name="mynode" />
  <test test-name="test_mynode" pkg="mypkg" type="test_mynode" />
</launch>
```

6.4 Integración Continua

El desarrollo de aplicaciones en ROS se realiza a menudo de forma colaborativa lo que supone que después de cada nuevo cambio en el código, haya que verificarlo para asegurar que no se introducen nuevos errores o se rompe la funcionalidad ya existente. Esto impulsa la necesidad de disponer de una infraestructura compartida que permita automatizar la ejecución de los tests de código después de cada nuevo cambio, así como también, permita compartir de forma eficiente los resultados de estos tests y las métricas entre los colaboradores.

La integración continua es el nombre que se le da a la automatización de las labores de compilación del código y ejecución automática de tests. Esta práctica se asocia, a menudo, con las metodologías de programación extrema y desarrollo ágil.

Normalmente, los diferentes pasos del flujo de trabajo de la integración continua se recogen en un archivo, que dependerá de la herramienta utilizada, y que se puede después ejecutar desde cualquier equipo. Sin embargo, la tarea de montar un servidor compartido de integración continua que permita mantener un histórico de los resultados de las compilaciones, automatizar la ejecución de los tests, compartir los resultados entre los desarrolladores, etc., es algo más complicado. Por ello, existen múltiples alternativas de lo que se denominan Servidores CI, cuya función es proporcionar estas utilidades en una plataforma propia. Entre los diferentes servidores CI que existen se pueden encontrar algunos de los más utilizados en proyectos basados en ROS[34]:

- Jenkins
- TravisCI
- GitLab CI
- Semaphore
- CircleCI
- Shippable

En este trabajo se va a hacer hincapié en el servidor TravisCI que es el que ha sido utilizado para llevar a cabo la integración continua a largo del desarrollo de este paquete.

El funcionamiento básico de Travis CI consiste en clonar el repositorio a un nuevo entorno virtual y llevar a cabo una serie de tareas para compilar y probar el código. Si alguna de las tareas falla, el 'build' se considera

[34] http://wiki.ros.org/CIs

fallido (‘broken’), mientras que si ninguna tarea falla, el ‘build’ se considera aprobado (‘passed’).

Como se puede ver, dentro del entorno de TravisCI, existen algunas palabras clave que tienen un significado específico:

- ‘job’: es un proceso automatizado que clona el repositorio en un entorno virtual y realiza una serie de ‘phases’ como compilar el código, ejecutar tests, etc. Un ‘job’ falla si el código que devuelve la fase ‘script’ no es cero.
- ‘phase’: es el conjunto de pasos secuenciales de un ‘job’. En TravisCI, existen las siguientes fases:
    - Install apt addons (opcional)
    - Install cache components (opcional)
    - before_install: instalar paquetes del SO necesarios para la construcción del proyecto.
    - install: instalar las dependencias (librerías) que sean necesarias para la construcción del proyecto.
    - before_script
    - script: ejecutar el script de construcción del proyecto.
    - before_cache (opcional): para limpiar la caché
    - after_success or after_failure: after_success se ejecutará si la construcción se realiza correctamente. after_failure se ejecutará en caso de que la construcción falle.
    - before_deploy (opcional)
    - deploy (opcional): desplegar la aplicación a una plataforma integrada con Travis CI como Heroku o Engine Yard.
    - after_deploy (opcional)
    - after_script
- ‘build’: es un grupo de ‘jobs’. Por ejemplo, un ‘build’ puede tener dos ‘jobs’, cada uno prueba un proyecto con una versión distinta, por ejemplo, de Java. Un ‘build’ termina cuando todos sus ‘jobs’ terminan. Una ‘build’ se considera que ha fallado si uno o más de sus ‘jobs’ no ha terminado correctamente. Esto puede ocurrir por una de estas tres razones:
    - ‘errored’: un comando en la fase before_install, install o before_script falló. En este caso el job termina inmediatamente.
    - ‘failed’: un comando en la fase script falló. El ‘job’ se ejecuta hasta que se termina
    - ‘canceled’: un usuario cancela el ‘job’ antes de que complete.
- ‘stage’: es un grupo de ‘jobs’ que se ejecutan en paralelo.

Travis CI ofrece tres tipos de infraestructuras diferentes:

- Basada en contenedores, que es la opción por defecto. Se ejecuta más rápido que la basada en máquinas virtuales, pero no soporta el uso de sudo, setuid o setgid.
- Basada en máquinas virtuales. Es un poco más lenta que la basada en contenedores para iniciarse, pero tiene más recursos.
- OS X. Para probar software que necesite el entorno OS X para funcionar.

Toda la información sobre cómo se debe realizar cada ‘job’ se especifica en un fichero “.travis.yml” que debe estar en la carpeta raíz del repositorio en cuestión. En su forma más básica, este fichero debe especificar el lenguaje utilizado en el proyecto para que cargue las librerías adecuadas en el entorno de ejecución y lo que se debe hacer en cada una de las fases descritas anteriormente. No es necesario especificar acciones en todas las fases.

A modo de ejemplo se expone el siguiente fichero “.travis.yml” :

```
language: python
python:
- "3.6"
env:
- DJANGO=2.0 DB=postgres
before_install:
- cd decide
install:
- pip install -r ../requirements.txt
script:
- python manage.py test
...
```

En este caso, se especifica que se quiere utilizar la versión 3.6 de Python y se indican un par de variables de entorno y los comandos que son necesarias para el funcionamiento de la aplicación:

- Moverse al path adecuado antes de la fase install.
- Instalar las dependencias que necesita el proyecto para funcionar en la fase install.
- Lanzar las pruebas en la fase script.

Gracias al uso de este servidor público de integración continua, todo el mundo, no solo los desarrolladores, tienen acceso a los resultados de compilación y tests de los repositorios públicos registrados. Esto permite al resto de usuarios conocer el nivel de rigurosidad y, por tanto, la probabilidad de que el paquete contenga más o menos fallos.

6.4.1 Ejecución automática de los tests

La ejecución automática de tests para un paquete de ROS, se lleva a cabo mediante la herramienta de compilación de ROS, es decir, catkin, incluyendo los tests en el archivo "CMakeLists.txt"[35][36]:

- Para incluir tests unitarios:

```
if (CATKIN_ENABLE_TESTING)
  catkin_add_nosetests(test/test_your_node.py
                       DEPENDENCIES ${catkin_EXPORTED_TARGETS}
                       DEPENDENCIES ${${PROJECT_NAME}_EXPORTED_TARGETS})
endif()
```

- Para incluir tests de integración:

```
if(CATKIN_ENABLE_TESTING)
  find_package(rostest REQUIRED)
  add_rostest(test/mytest.test)
endif()
```

Finalmente se pueden ejecutar mediante el comando:

```
$ catkin_make run_tests
```

[35] http://wiki.ros.org/rostest/Writing
[36] http://docs.ros.org/en/melodic/api/catkin/html/howto/format2/index.html

6.5 Cobertura de código

Es bien sabido ya que los tests mejoran la calidad del código y la previsibilidad de las versiones del software, sin embargo, ¿qué tan bien prueban los tests realmente el código? ¿cuántos tests son suficientes? Estas son las preguntas que busca responder la medición de la cobertura de código.

La cobertura de código ('code coverage') es una medida que se utiliza para describir el grado en que se ejecuta el código fuente de un programa cuando se realizan una serie de tests en particular, proporcionando un índice de calidad al conjunto de estos tests[29]. La medición simplemente determina qué declaraciones en un cuerpo de código se han ejecutado en el desarrollo del test y qué declaraciones no, de tal manera que un programa con una alta cobertura, es decir, que hay una mayor parte de su código fuente que ha sido ejecutado durante el test, indica una probabilidad menor de contener errores de software no detectados en comparación con un programa con una cobertura de código más baja.

En general, un sistema de cobertura de código recopila información sobre el programa en ejecución y luego la combina con la información de origen para generar un informe sobre la cobertura de código del conjunto de tests, culminando así un ciclo de retroalimentación en el proceso de desarrollo: a medida que se desarrollan los tests, la cobertura resalta aspectos del código que pueden no ser probados adecuadamente y que requieren pruebas adicionales. Este ciclo continúa hasta que la cobertura alcanza un objetivo específico.

Esta medición ayuda también a evitar la tendencia a que los tests se atrofien conforme el código pasa por múltiples ciclos de lanzamiento. A medida que se agrega nuevo código, es posible que no cumpla con los mismos estándares de prueba que se propusieron cuando se lanzó el proyecto por primera vez. Medir la cobertura del código puede mantener los tests a la altura de los estándares que necesita.

En resumen, se mide la cobertura del código por las siguientes razones:

- Para saber qué tan bien los tests prueban realmente el código.
- Para saber si hay suficientes tests.
- Para mantener la calidad de los tests durante el ciclo de vida de un proyecto.

Para calcular el índice de cobertura del código que ha sido ejecutado, que generalmente se mide en valores porcentuales (cuanto más cerca del 100%, mejor), las herramientas utilizan uno o más criterios que sirven para determinar cómo se ejecutó o no el código durante la realización del conjunto de tests. Estos criterios generalmente se definen como reglas o requisitos que el conjunto de tests deben cumplir. Entre los principales se pueden incluir:

- Cobertura de funciones: cuántas funciones definidas se han llamado.
- Cobertura de declaraciones: cuántas declaraciones del programa se han ejecutado.
- Cobertura de ramas: cuántas de las ramas de las estructuras de control se han ejecutado.
- Cobertura de condiciones: cuántas de las subexpresiones booleanas se han probado para un valor verdadero y falso.
- Cobertura de línea: cuántas líneas de código fuente se han probado.

Para satisfacer estas necesidades se han creado numerosas herramientas, tanto de código abierto como comerciales y que estas están disponibles para muchos lenguajes de programación incluso formando parte de herramientas de control de calidad: en herramientas de compilación como Ant, Maven y Gradle, en herramientas de CI como Jenkins, en herramientas de gestión de proyectos como Jira, etc. Algunos ejemplos de herramientas de cobertura de código son:

- Coverage.py (Python)
- JCov (Java)
- Bullseye Coverage (C y C++)
- etc.

Por otro lado, más allá de la herramienta que se usa para medir la cobertura, al igual que para la integración

continua, existen servicios web que permiten mantener un seguimiento de esta métrica, proveyendo una plataforma común entre los desarrolladores:

- Codecov
- Code Climate
- Codacy
- Coveralls
- Entre otros

En este paquete se ha utilizado la herramienta Coverage.py integrada en el servicio web Codecov para llevar el seguimiento de la cobertura del conjunto de tests. Esta plataforma proporciona algunas herramientas de visualización en forma de gráficos que permiten analizar los resultados obtenidos de forma más intuitiva, como los que se muestran en la siguiente imagen.

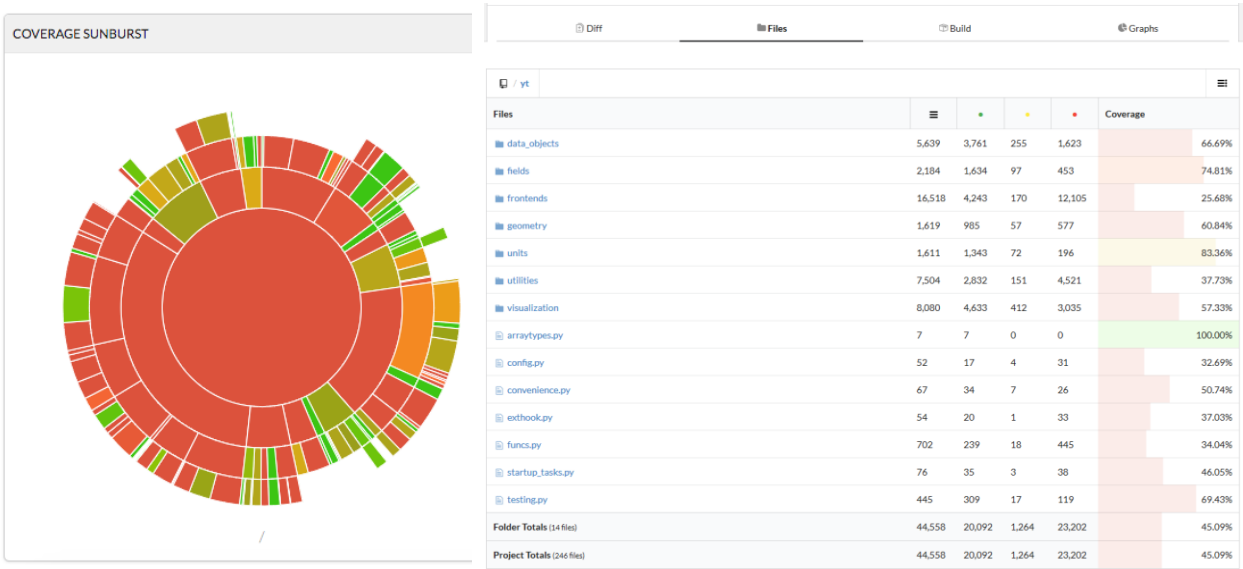


Figura 6-2. Ejemplo de muestra de los resultados de cobertura de código en la plataforma Codecov. A la izquierda se muestran los resultados en un gráfico estilo 'sunburst', y a la derecha se muestran en forma de valores porcentuales para cada archivo.

6.6 Licencia

Como ya se ha dicho, ROS es un proyecto de código abierto cuyo objetivo es apoyar a una amplia variedad de usuarios y desarrolladores, desde estudiantes graduados hasta empresarios, por lo que se prefieren las licencias permisivas de código abierto que facilitan el uso comercial del código.

La licencia preferida para el proyecto es la licencia BSD, aunque cualquier licencia aprobada por la OSI es aceptable. En el caso de este proyecto se usa la licencia Apache 2.0[37] que es una licencia de software libre creada por la Apache Software Foundation (ASF). La licencia requiere la conservación del aviso de copyright y el 'disclaimer', y es una licencia sin 'copyleft', es decir, no requiere la redistribución del código fuente cuando se distribuyen versiones, solo exige que se mantenga una nota que informe a los usuarios que en la distribución se ha usado código con la Licencia Apache. Esta propiedad cumple, además, con las recomendaciones de ROS, con el fin de que los archivos autogenerados y las estructuras de datos, como los mensajes (.msg) y los servicios no se vean obstaculizados.

El texto completo con las declaraciones de la licencia Apache 2.0, así como la de cualquier otra, debe ser colocado en el directorio LICENSES en la raíz del repositorio. Además, cada archivo fuente debe contener un

[37] https://www.apache.org/licenses/LICENSE-2.0

resumen comentado de la licencia en la parte superior.

```
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Como comentario adicional respecto al tema de licencia de respositorios, hay que tener en cuenta que se deben obedecer rigurosamente los términos de las licencias de software de terceros que se hayan utilizado en el paquete. Por ejemplo, si el paquete usa una biblioteca GPL, entonces el código del paquete también debe ser estar licenciado con GPL.

6.7 Documentación

Se puede decir que los principales enfoques de la documentación de cualquier aplicación software son: el desarrollo, el mantenimiento y la transferencia de conocimiento a otros desarrolladores.

Una documentación confiable y fácilmente accesible es siempre imprescindible y juega un papel muy importante a la hora de determinar el éxito de la aplicación, ya que simplifica y acelera el proceso de aprendizaje de los nuevos usuarios, ayuda a reducir los costes de soporte, etc., en conclusión, mejora la calidad del producto final.

Desde el punto de vista de los paquetes de ROS, se pueden encontrar dos tipos de documentación en línea principalmente:

- Wiki del paquete: proporciona información a alto nivel del paquete centrándose fundamentalmente en las APIs a nivel de ROS (topics, servicios, parámetros). Está orientado a personas que ejecutan el código.
- Code API: se trata de la documentación del código fuente y está orientado a personas que integran o editan el código.

6.7.1 ROS Wiki del paquete

La página Wiki de un paquete de ROS se genera conforme a una guía de estilo[38] utilizando las wiki-markups[39] y las WikiMacros[40] para obtener los efectos de formato especiales. Aunque no es obligatoria, existe una plantilla[41] propuesta que establece la información del paquete que se debe incluir y la estructura que se debe seguir:

1) Resumen del paquete: debe resumir qué funcionalidad exporta y a qué nivel (es decir, biblioteca C++, scripts, nodos ROS, etc.).

[38] http://wiki.ros.org/StyleGuide
[39] http://wiki.ros.org/HelpOnEditing
[40] http://wiki.ros.org/WikiMacros
[41] http://wiki.ros.org/StackDocumentation

2) Documentación de la API de ROS incluyendo:
   - Configuraciones de nodos/grafos. Para cada archivo ".launch" una imagen graphviz de los nodos y sus conexiones.
   - Lista de frames tf utilizados.
   - Lista de topics utilizados.
   - Lista de servicios utilizados.
   - Lista de parámetros utilizados.
3) Lista de paquetes (autogenerados).
4) Aplicaciones relacionadas, es decir, dónde puede el usuario encontrar usos interesantes del paquete (si procede).
5) Dependencias (autogeneradas).
6) " Click here to file a bug report"
7) Changelist
8) Hardware
9) Tutoriales de uso.
10) Videotutoriales
11) Depuración y Visualización. Descripción de cómo visualizar el estado del paquete y depurar problemas.

Siguiendo estas pautas, en la imagen siguiente se muestra, a modo ilustrativo, el resultado de una de las Wikis generadas para el paquete Z_LASER_PROJECTOR.

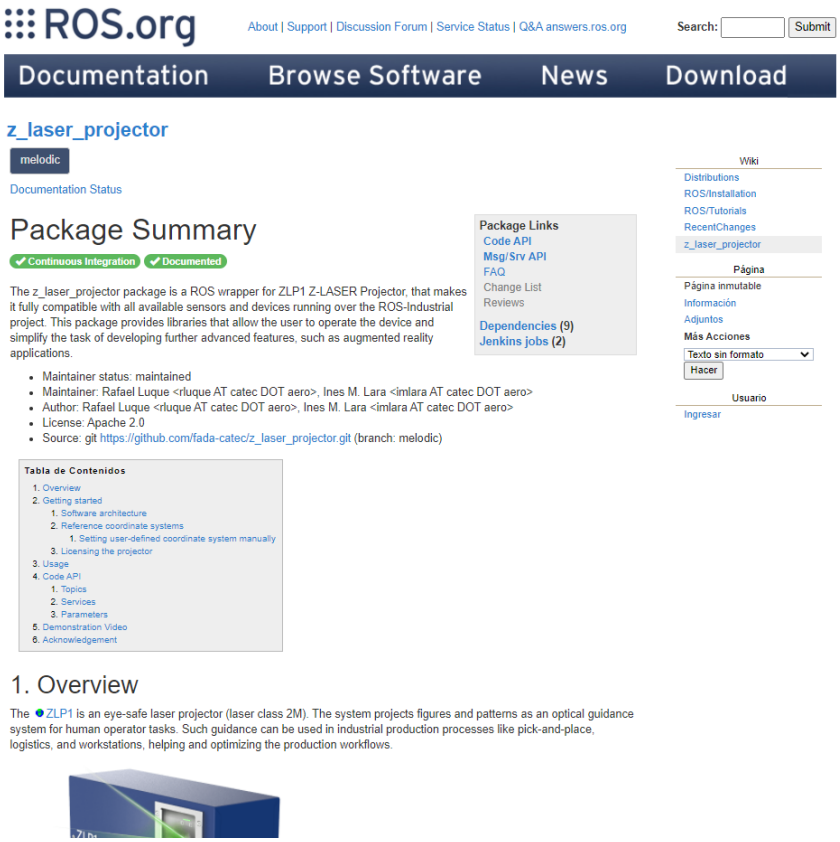


Figura 6-3. Página Wiki del paquete Z_LASER_PROJECTOR.

6.7.2 Code API

Por otro lado, se encuentra la documentación relacionada con el código fuente o también llamada Code API. Para esta tarea existen algunas herramientas que permiten generar automáticamente la documentación directamente a partir de comentarios incluidos en los archivos de código. Concretamente para los desarrollos basados en ROS, se utiliza el paquete rosdoc_lite que se encarga de ejecutar alguna de las siguientes herramientas externas de documentación de código: Doxygen[42], para código escrito en lenguaje C++, y Epydoc[43] o Sphinx[44], para documentar código escrito en lenguaje Python. En general, estas herramientas lo que hacen es buscar comentarios estructurados en todo el código fuente y generar archivos HTML en los que se refleja esta información.

Para usar este paquete, una vez ya instalado, se utiliza un archivo de configuración rosdoc basado en YAML. Este archivo lo que permite es: usar Epydoc o Sphinx en lugar de Doxygen que es el que se ejecuta por defecto; ejecutar múltiples herramientas de documentación por paquete; establecer configuraciones avanzadas para la documentación. Este archivo, "rosdoc.yaml", se incluye en la carpeta raíz del paquete en el que se indica, como se puede ver en el siguiente código, el tipo lenguaje de código que se quiere documentar y el directorio donde se quiere almacenar la documentación generada, además de otras opciones que también se admiten:

```
- builder: sphinx
  sphinx_root_dir: doc
```

Finalmente, se habilita este archivo de configuración rosdoc incluyéndolo en el "package.xml":

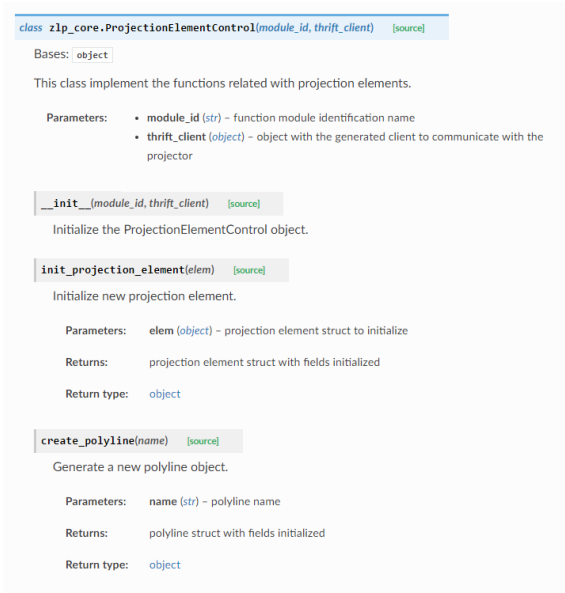
```
<export>
  <rosdoc config="rosdoc.yaml" />
</export>
```

Para cada una de las herramientas mencionadas anteriormente, la estructura de los comentarios dentro del código es diferente e incluso se aceptan varios tipos. En el caso de Sphinx, que es la herramienta que aquí se utiliza, los comentarios en el código se han descrito siguiendo la estructura Google Style Docstring[45]. A modo de ejemplo, se expone a continuación la estructura de un comentario para documentar una función con sus argumentos:

```
def function_with_types_in_docstring(param1, param2):
    """Example function with types documented in the docstring.
    Args:
        param1 (int): The first parameter.
        param2 (str): The second parameter.
    Returns:
        bool: The return value. True for success, False otherwise.
    """
```

Como ya se ha comentado, partir de estos comentarios, la herramienta genera los archivos en formato XML que incluyen toda la documentación.

[42] http://wiki.ros.org/Doxygen
[43] http://wiki.ros.org/Epydoc
[44] http://wiki.ros.org/Sphinx
[45] https://google.github.io/styleguide/pyguide.html



class zlp_core.ProjectionElementControl(module_id, thrift_client) [source]

Bases: object

This class implement the functions related with projection elements.

Parameters:
- module_id (str) – function module identification name
- thrift_client (object) – object with the generated client to communicate with the projector

__init__(module_id, thrift_client) [source]

Initialize the ProjectionElementControl object.

init_projection_element(elem) [source]

Initialize new projection element.

Parameters: elem (object) – projection element struct to initialize

Returns: projection element struct with fields initialized

Return type: object

create_polyline(name) [source]

Generate a new polyline object.

Parameters: name (str) – polyline name

Returns: polyline struct with fields initialized

Return type: object

Figura 6-4. Ejemplo de CODE API del código del paquete Z_LASER_PROJECTOR utilizando la herramienta Sphinx y el estilo Google Docstring.

6.8 Lanzamiento

El lanzamiento (‘release’) de un paquete en ROS hace referencia a la liberación de los archivos binarios generados tras una correcta compilación del paquete que permiten su instalación directamente con alguna herramienta de gestión de paquetes (APT), propia de cada sistema operativo. En el caso de Ubuntu se podría instalar el paquete directamente con el siguiente comando:

```
$ apt-get install ros-rosdistro-example-package
```

Sin embargo, esto se dejará para más adelante ya que antes del lanzamiento del paquete hay que indexarlo al repositorio de ROS Index[46][47]. Este paso es necesario para que el paquete esté disponible públicamente y sea visible para los demás desarrolladores. Para llevar a cabo esta esta operación se siguen una serie de pasos[48] que se deben repetir para cada distribución de ROS en la que se quiera lanzar el paquete, garantizando así que el paquete ha sido compilado y probado en esa versión específica.

De forma resumida, este conjunto de pasos consiste en solicitar (mediante pull-request) que el paquete desarrollado se añada al listado de paquetes incluidos en el archivo “distribution.yaml” del paquete rosdistro, en la distribución de ROS que corresponda, por ejemplo, “rosdistro/melodic/distribution.yaml”. En esta petición, lo que se solicita es incluir el siguiente trozo de código en el archivo “distribution.yaml” mencionado.

```
example-package:
    source:
        type: git
        url: https://github.com/my-example-org/example-package.git
        version: master
    status: maintained
```

Tras confirmarse la petición:

[46] https://index.ros.org/
[47] http://wiki.ros.org/rosdistro/Tutorials/Indexing%20Your%20ROS%20Repository%20for%20Documentation%20Generation
[48] https://index.ros.org/contribute/

- El paquete se sube al repositorio de ROS Index.
- La página wiki se enlaza con el paquete y extrae automáticamente la información del "package.xml".
- La información extraída ayuda a los motores de búsqueda de Internet a indexar la página y conduce a una mejor visibilidad.

Figura 6-5. Ejemplo de página Wiki enlazada al repositorio del paquete una vez se ha añadido al ROS Index.

Ahora sí, una vez el paquete ha sido añadido correctamente al ROS Index, es el turno del lanzamiento del paquete en Debian. Para ello, se utiliza bloom[49], una herramienta que permite automatizar el lanzamiento a partir del código fuente[50].

A modo de recopilatorio, se enumeran las herramientas, de las detalladas en esta sección, que han sido utilizadas para garantizar la calidad del software de este proyecto: Git para el control de versiones; las guías de estilo de ROS y Python; Pylint como escáner de código; unittest, rosunit y rostest para los tests unitarios y los tests de integración; Travis-CI para la integración continua; Coverage.py y Codecov para la cobertura del código; Apache 2.0 para licenciar el paquete; y, por último, rosdoc_lite y Sphinx para documentar el código fuente, siguiendo el estilo Google docstring.

[49] http://wiki.ros.org/bloom
[50] http://wiki.ros.org/bloom/Tutorials/FirstTimeRelease

7 CONCLUSIONES

La realización de este trabajo contribuye al crecimiento de la comunidad de ROS en el ámbito industrial, ampliando el horizonte de desarrollo de nuevas aplicaciones en un campo como es el de la realidad aumentada, a la par que se atrae la atención de nuevos fabricantes.

Al mismo tiempo, se trata de un paquete donde se garantiza la calidad del software mediante un gran abanico de herramientas que, aunque presentan una curva de aprendizaje lenta, una vez se familiariza con ellas y se adquiere agilidad, reducen enormemente los esfuerzos posteriores de mantenimiento (corregir errores, mejorar el rendimiento, etc.).

Cabe decir que el uso de Python 3 como lenguaje de programación del código fuente ha presentado algunos inconvenientes a la hora de compilar el proyecto debido a problemas de compatibilidad entre diferentes paquetes de la distribución "melodic" de ROS. El motivo por el que se ha utilizado este lenguaje y en esta versión es debido al módulo Thriftpy que ha sido necesario importar para utilizar la interfaz de Apache Thrift en la comunicación con el dispositivo de proyección.

Como posibles mejoras previstas para este paquete se encuentra el desarrollo de otras funcionalidades como la proyección de figuras a partir de archivos gráficos en formato HPGL, que es el que reconoce el dispositivo. Además, como este formato no es el estándar, también se pretende crear un convertidor de archivos gráficos entre un formato que sí sea estándar, como el DXF o DWG, y este que no lo es, HPGL. Otra mejora que también se puede incorporar es la proyección de objetos en 3D que el proyector también es capaz de generar.

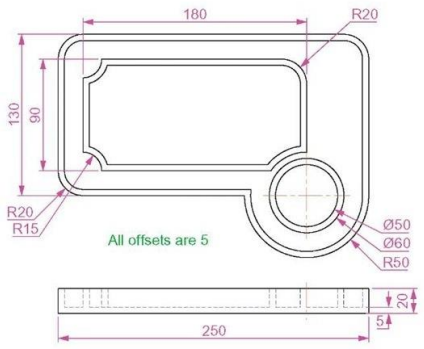


Figura 7-1. Ejemplo de figura en archivo gráfico con formato DWG.

Para terminar. se expone un ejemplo de solución basada en realidad aumentada con proyección láser en la que se ha utilizado este modelo de proyector para desarrollar la aplicación. Se trata del proyecto AEROPAINT, un proyecto desarrollado en CATEC y que está orientado a la logística y robotización de los procesos de pintura de piezas aeronáuticas. En este proyecto se integran diversas tecnologías con el fin de obtener un proceso automatizado de pintado y verificación de componentes aeroespaciales con alta productividad, eficiencia y respeto por el medioambiente.

El proceso está formado por dos fases, una primera fase en la que la que se realiza la inspección de la pieza en busca de defectos de pintura utilizando para ello técnicas de inteligencia artificial, y una segunda fase consistente en la ubicación de los defectos mediante proyección láser con el objetivo de indicar al operario sus posiciones. Este proyecto supone un ejemplo de implementación de procesos interconectados en el camino hacia una fábrica digital y conectada.

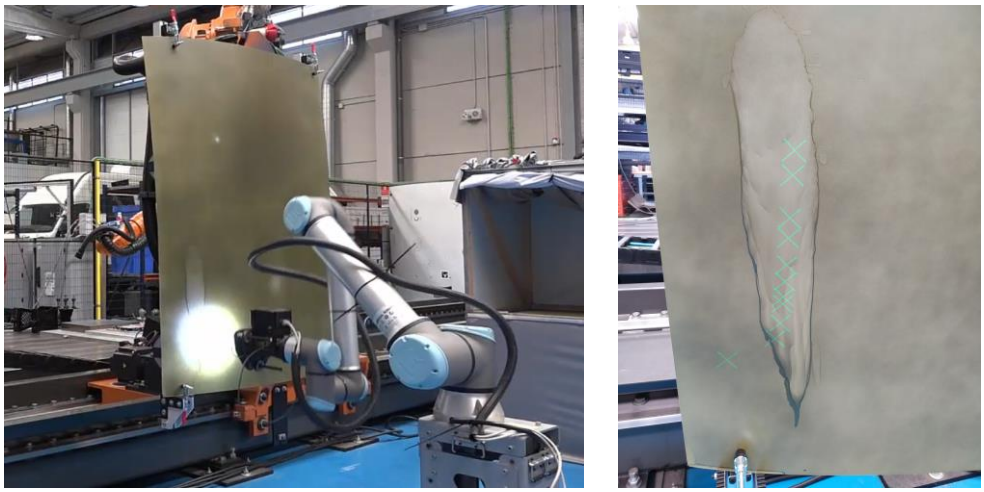


Figura 7-2. Aplicación de realidad aumentada basada en proyección láser. Proyecto AEROPAINT (FADA-CATEC).

REFERENCIAS

[1] Annex K. Actions involving financial support to third parties. HORIZON 2020 – WORK PROGRAMME 2018-2020. General Annexes. Extraído de: https://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2018-2020/annexes/h2020-wp1820-annex-k-fs3p_en.pdf

[2] (2019). ROSIN Call for Focused Technical Projects: Applicant Guide. Extraído de: https://www.rosin-project.eu/

[3] Ronald T. Azuma. (1997). A survey of augmented reality. Presence: Teleoperators and Virtual Environments 6, 4, 355–385.

[4] Milgram, Paul & Kishino, Fumio. (1994). A Taxonomy of Mixed Reality Visual Displays. IEICE Trans. Information Systems. vol. E77-D, no. 12. 1321-1329.

[5] Billinghurst, Mark & Clark, Adrian & Lee, Gun. (2015). A Survey of Augmented Reality. Foundations and Trends in Human-Computer Interaction. 8.

[6] Steve Aukstakalnis. 2017. Practical augmented reality: a guide to the technologies, applications, and human factors for AR and VR. Boston: Addison-Wesley

[7] Van Krevelen, D. W. F., & Poelman, R. (2010). A Survey of Augmented Reality Technologies, Applications and Limitations. The International Journal of Virtual Reality, 9(2), 1-20.

[8] Matthew S. Brennesholtz. (2006). Understanding 3LCD Technology. Part of a continuing series of technical tutorials from NSCA and Pro AV Magazine.

[9] Rolland, Jannick & Fuchs, Henry. (2000). Optical Versus Video See-Through Head-Mounted Displays in Medical Visualization. Presence. 9. 287-309.

[10] Mekni, M. and Lemieux, A. (2014). Augmented Reality: Applications, Challenges and Future Trends. Computer and applied computational science.

[11] Kim, Soo-Kyun & Kang, S.-J & Choi, Yoo-Joo & Choi, M.-H & Hong, Min. (2017). Augmented-Reality Survey: from Concept to Application. KSII Transactions on Internet and Information Systems. 11. 982-1004.

[12] Carmigniani, J., & Furht, B. (2011). Augmented Reality: An Overview. Handbook of Augmented Reality, 3–46.

[13] Nee, Andrew & Ong, S K & Chryssolouris, George & Mourtzis, Dimitris. (2012). Augmented reality applications in design and manufacturing. CIRP Annals - Manufacturing Technology. 61. 657–679.

[14] M. Bruehl. (2005). Laser template projection for composite assembly. Reinforced Plastics, Volume 49, Issue 9, Pages 56-57.

[15] D. Arkell. (2008). The guiding light. New laser-guided tool boosts efficiency and accuracy at the Everett paint hangar. Boeing Frontiers. Commercial Airplanes.

[16] Himperich, Frederick. (2007). Applications of Augmented Reality in the Automotive Industry.

[17] K. Pentenrieder, C. Bade, F. Doil and P. Meier. (2007). Augmented Reality-based factory planning - an application tailored to industrial needs. 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 31-42.

[18] Doshi, Ashish & Smith, Ross & Thomas, Bruce & Bouras, Con. (2017). Use of projector based

augmented reality to improve manual spot-welding precision and accuracy for automotive manufacturing. The International Journal of Advanced Manufacturing Technology. 89.

[19] (2020). Datasheet ZLP1. Z-LASER. Extraído de: https://z-laser.com/en/product/laser-projector/zlp1/

[20] Agarwal, A., Slee, M. & Kwiatkowski, M. (2007). Thrift: Scalable Cross-Language Services Implementation. Facebook.

[21] Randy Aberneth. (2019). Programmer's Guide to Apache Thrift. Manning Publications. 592 pag.

[22] Hauge Ø., Sørensen CF., Conradi R. (2008) Adoption of Open Source in the Software Industry. In: Russo B., Damiani E., Hissam S., Lundell B., Succi G. (eds) Open Source Development, Communities and Quality. OSS 2008. IFIP – The International Federation for Information Processing, vol 275. Springer, Boston, MA.

[23] Nagy, Del & Bhattacherjee, Anol. (2010). Organizational adoption of open source software: Barriers and remedies. Commun. ACM. 53. 148-151.

[24] International Standard ISO: ISO/IEC 25010–2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. International Organization for Standardization ISO (2011)

[25] Adam Alami. (2020). The sustainability of quality in free and open source software. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings (ICSE '20). Association for Computing Machinery, New York, NY, USA, 222–225.

[26] Halloran, T. & Scherlis, William. (2002). High Quality and Open Source Software Practices.

[27] Michlmayr, Elke. (2005). A case study on emergent semantics in communities. In Workshop on Semantic Network Analysis, International Semantic Web Conference (ISWC2005).

[28] A. Alami, Y. Dittrich and A. Wasowski. (2018). Influencers of Quality Assurance in an Open Source Community. IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), Gothenburg, 2018, pp. 61-68

[29] H. Hemmati. (2015). How Effective Are Code Coverage Criteria?. IEEE International Conference on Software Quality, Reliability and Security, Vancouver, BC, pp. 151-156.

GLOSARIO

2D: Bidimensional

3D: Tridimensional

7PM: Séptimo Programa Marco de Investigación e Innovación

API: Application Programming Interface

APT: Advanced Packaging Tool

ASF: Apache Software Foundation

BSD: Berkeley Software Distribution

CAGR: Compound Annual Growth Rate

CATEC: Centro Avanzado de TECnologías aeroespaciales

CI: Continuous Integration

CVCS: Centralized Version Control System

DXF: Drawing Exchange Format

DVCS: Distributed Version Control System

DWG: Drawing

FADA: Fundación Andaluza para el Desarrollo Aeroespacial

FI-PPP: Asociación Público-Privada de Internet del Futuro

FSTP: Financial Support to Third Parties

FOV: Field Of View

FTP: Focused Technical Projects

GPS: Global Positioning System

GUI: Graphical user interface

H2020: Horizon 2020

HMD: Head Mounted Display

HMI: Interfaz Hombre-Máquina

HPGL: Hewlett-Packard Graphics Language

I+D+i: Innovación, desarrollo e investigación

IDE: Integrated Development Environment

IDL: Interactive Data Language

IP Code: Ingress Protection Code

LEA: Límite de Emisión Accesible

OSI: Open Systems Interconnection

OSS: Open Source Software

OSRF: Open Source Robotics Foundation

OST: Optical See-Trough

PC: Personal Computer

PYME: Pequeña y Mediana Empresa

QA: Quality Assurance

RA: Realidad Aumentada

REST: Representational State Transfer

RPC: Remote Procedure Call

ROS: Robotic Operative System

ROSIN: ROS-Industrial quality-assured robot software components

RV: Realidad Virtual

STAIR: STanford AI Robot

TIC: Tecnologías de la Información y la Comunicación

UAV: Unmanned Aerial Vehicle

UE: Unión Europea

UGV: Unmanned Ground Vehicle

VA: Virtualidad Aumentada

VCS: Version Control System

VST: Video See-Through

XML: Extensible Markup Language