Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Operación de remachado mediante robot manipulador industrial basado en ROS

Autor: Víctor Hugo Gómez Tejada
Tutor: Guillermo Heredia Benot

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

Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnología Industriales

Operación de remachado mediante robot manipulador industrial basado en ROS

Autor:
Víctor Hugo Gómez Tejada

Tutor:
Guillermo Heredia Benot
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Proyecto Fin de Grado: Operación de remachado mediante robot manipulador industrial basado en ROS

Autor: Víctor Hugo Gómez Tejada

Tutor: Guillermo Heredia Benot

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

Resumen

El diseño e implementación de procesos cada vez más automatizados en la industria ha empujado el desarrollo tecnológico del campo de la robótica en los últimos años. Esto ha propiciado la iniciativa de promover e invertir en el campo de investigación de la robótica industrial.

EuRoC, una organización de carácter europeo, ha promulgado la iniciativa en la investigación en este campo mediante el lanzamiento de un desafío estructurado en distintos niveles, aquí es donde se encuentra el proyecto en cuestión.

La organización nos plantea un reto de carácter industrial, donde nos situa en un entorno de simulación donde debemos realizar una tarea de remachado a una pieza de trabajo mediante un robot manipulador industrial. En este desafío se nos valorará la capacidad para programar nuestro robot mediante el framework robótico ROS para realizar la tarea de remachado, evaluando su rapidez y eficiencia.

En las siguientes páginas se presentará el proyecto, así como su solución propuesta con su calificación final por parte del colectivo EuRoC.

Índice

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

El proyecto que presento tiene como contexto el concurso de Robótica a nivel Europeo de la organización EuRoC, desarrollada en el Área de Automatización y Robótica del centro de investigación CATEC. Dentro del concurso, participamos en el primer Challenge, dónde se proponen soluciones a una situación estándar de trabajo de interacción operario-robot.

Dentro del primer Challenge, el proyecto está situado en la primera fase, dónde se nos proporciona un entorno de simulación facilitado por la organización para implementar soluciones al desafío lanzado.

Esta primera fase consta de dos escenarios distintos. El primero, donde se desarrolla mi proyecto, tiene lugar entre un robot manipulador y su interacción con una pieza de trabajo, y el segundo se trata de un desafío de cooperación entre dos brazos robóticos.

En esta fase, se me encomienda una tarea de aplicación industrial: Proceso de remachado de los agujeros de una pieza de trabajo. La finalidad de esta tarea es la programación del robot manipulador que nos proporciona la organización (configuración UR5) para realizar la misión citada anteriormente: Remachado de la pieza a través de nuestro robot manipulador.

La solución del problema planteado será evaluada por la organización EuRoC en busca de la mayor eficiencia y seguridad posible, es decir, en un tiempo establecido, se evaluará el número de remachados realizados con éxito, realizando movimientos suaves con el robot manipulador sin tocar la pieza de trabajo garantizando así un entorno seguro y sin riesgos laborales.

1.1 EuRoC

EuRoC (European Robotic Challenge) [1] consiste en tres desafíos de robótica en la industria relevante destinadas a estimular nuevas innovaciones para la industria de manufacturación europea.

Figura 1: Logo EuRoC.

El consorcio EuRoC se compone de un total de 9 beneficiarios: 5 académicos/investigación, 3 compañías y una PYME.

Los cinco beneficiarios académicos/investigación (CREATE, CNRS, DLR, ETHZ, IPA) del consorcio han sido elegidos de manera que los tres retos de EuRoC estén cubiertos en un alto nivel científico, mientras que los tres beneficiarios industriales (AIR, ASC, KUKA) garantizan la relevancia industrial de los retos en los ámbitos pertinentes que se cree para cubrir las tecnologías más prometedoras para los posibles usuarios finales impulsada por el mercado de la investigación robótica. Además, un beneficiario con una gran experiencia en estos desafíos (INNO) se ha incluido en el consorcio para agilizar los procesos, garantizar los profesionales de diseño, la ejecución de los desafíos y la comparabilidad entre los tres escenarios de cada uno de ellos.

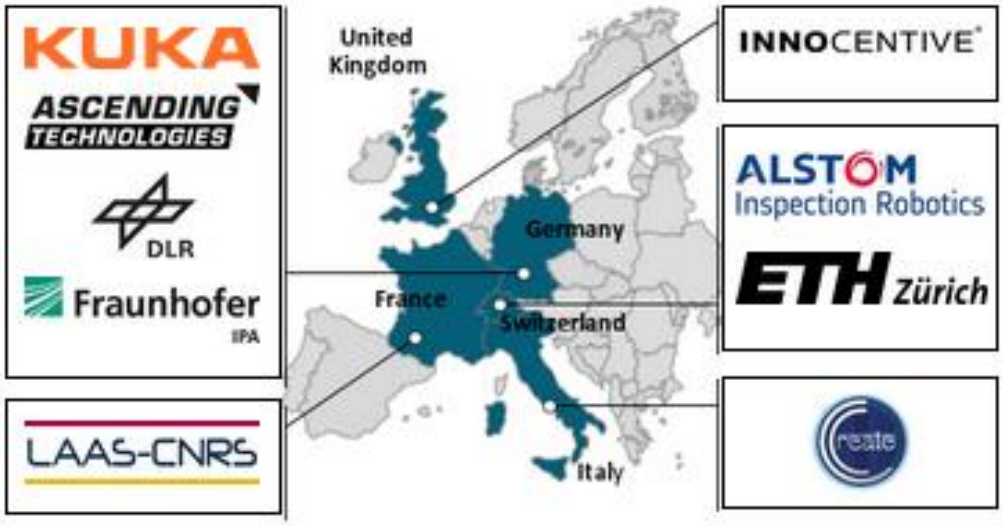

Figura 2: Consorcio EuRoC.

1.1.1 Resumen

La industria de manufacturación europea necesita soluciones competitivas para mantener el liderazgo mundial en productos y servicios. La explotación de sinergias a través de expertos en aplicaciones, proveedores de tecnología, integradores de sistemas y proveedores de servicios acelerará el proceso de llevar las tecnologías innovadoras de los laboratorios de investigación a los consumidores industriales finales. Para facilitar en este contexto, la iniciativa EuRoC propone poner en marcha tres retos de la industria relevantes:

- Reconfigurable Interactive Manufacturing Cell (RIMC).
- Shop Floor Logistics and Manipulation (SFLM).
- Plant Servicing and Inspection (PSI).

El objetivo es afilar el foco de la industria europea a través de una serie de experimentos de aplicación, mientras adoptamos un enfoque innovador que garantice la evaluación del desempeño comparativo. Cada desafío se lanza a través de una convocatoria abierta, y se estructura en 3 etapas.

En cada desafío, que consta como ya hemos dicho de 3 etapas, 45 concursantes de seleccionan utilizando un desafío en un entorno de simulación: esta baja barrera de entrada permite a los nuevos jugadores competir con los equipos robóticos establecidos, en esta etapa, la primera, perteneciente al desafío Reconfigurable Interactive Manufacturing Cell (RIMC) es en el que participamos el equipo del área de Automática y Robótica de CATEC.

Tras esta primera etapa, 15 equipos serán admitidos para la segunda, donde se deberán formar un equipo estándar formado por expertos en investigación, proveedores de tecnología, integradores de sistemas, además de usuarios finales.

Los equipos están obligados al uso de estas plataformas robóticas de referencia facilitadas por este consorcio para esta segunda etapa. Después de una evaluación intermedia con el concurso público, los equipos avanzan a la última etapa para mostrar el caso desarrollado en un entorno realista.

Después de un proceso de evaluación abierto, 6 finalistas serán admitidos para ejecutar estos experimentos pilotos en un entorno real en los sitios de usuarios finales para determinar el ganador final de EuRoC.

1.1.2 Objetivos

Objetivo 1: Lanzamiento y ejecución de tres desafíos industrialmente relevantes

Un factor clave para impulsar la innovación en robótica y manufacturación de Europa es el fortalecimiento de la colaboración y el intercambio de ideas entre la industria y la comunidad investigadora. Hacia este objetivo se ponen en marcha y ejecutan tres desafíos industrialmente relevantes en la robótica europea con aplicabilidad en la fábrica del futuro. Estos desafíos cubren los escenarios de aplicación más prometedores en la industria robótica europea.

Objetivo 2: Apoderamiento de plataformas robóticas e infraestructuras de referencia

El segundo objetivo de este proyecto es capacitar a la robótica de plataformas e infraestructuras de referencia para permitir a los investigadores concentrarse en su investigación "desafiante" en lugar de perder tiempo con problemas de bajo nivel relacionados con diseño y mantenimiento. Asimismo, los entornos se establecerán de tal manera que permitan la comparación de los diferentes enfoques metodológicos para la resolución de casos de uso típicos de fabricación.
Además, se proporciona la funcionalidad de alto nivel en las áreas de percepción, planificación y control que permita a los aspirantes probar y validar su I+D en contextos significativos. Los componentes de software tienen la misma funcionalidad, pero diferente aplicación y rendimiento, por lo que serán intercambiables y, en definitiva, comparables. Con la concesión por parte de numerosos investigadores y desarrolladores de tecnología que dan acceso a estas plataformas y entornos de referencia y ofrecen apoyo intensivo a través de los anfitriones del concurso y los fabricantes de robots, se espera que se levanten unos niveles de rendimiento sin precedentes hacia el final de este proyecto. Además, existe la intención de proporcionar, con una configuración específica, acceso remoto completo y seguro para la programación de los robots en la web, así como para el uso de los recursos de computación distribuida.
Si este concepto demuestra ser viable y eficiente, se podrían abrir nuevas perspectivas y posibilidades para los retos del futuro aprovechando el potencial multiplicativo de las tecnologías de nube.

Objetivo 3: Sostenibilidad y adaptabilidad a los usuarios finales

El tercer objetivo de este proyecto es el desarrollo de soluciones sostenibles: las soluciones de aplicaciones desarrolladas se pondrán a prueba de forma continua. En primer lugar en entornos de simulación y posteriormente, en entornos reales. Las instalaciones se verán aumentadas en niveles de madurez a través del proyecto. Aunque las soluciones son impulsadas para necesidades del usuario final, éstas serán suficientemente generales para que puedan ser aplicadas a otros usuarios finales, tareas y situaciones, y permitir una rápida comercialización. La robustez tendrá que aumentar significativamente para conseguir que los robots trabajen de forma continuada. Siguiendo esta metodología, se definirán criterios de referencia claros en los procesos de evaluación. Los problemas que no puedan resolverse en el plazo de este proyecto se integrarán en el proceso de actualización de la robótica y los planes de trabajos futuros.

1.1.3 Challenge 1

El escenario de aplicación es abierto, para fomentar la innovación y creatividad en relación con el desarrollo de aplicaciones de fabricaciones novedosas que involucran la colaboración entre humanos y robots. Sin embargo, la viabilidad y rentabilidad de las soluciones deberán de ser demostradas. Además, la seguridad en el desarrollo de la solución para el trabajo tendrá que ser garantizada a través de las medidas apropiadas. Las aplicaciones pueden implicar un brazo, dos o múltiples configuraciones de brazo, donde cada uno de ellos es reconfigurable. Además, la distribución de las tareas asignadas al empleado humano y a los brazos robóticos también puede reconfigurarse dinámicamente en línea. Encontramos problemas tales como:

- Técnicas de percepción y cognición adaptativas en presencia de diseños complejos de células de trabajo y entornos dinámicos con cambios de iluminación y tolerancias.
- Técnicas de montajes robustos en presencia de piezas con tolerancias flexibles.
- Interacción humano-robot segura y productiva en presencia de situaciones ambiguas y lugares de trabajo reducidos.

- Métodos de control de sistemas robóticos de cooperación multi-función en un entorno industrial relevante.

En la primera fase de este challenge tiene lugar mi proyecto. Consta de 4 tareas interactivas robot-operario robot-robot, que se realizarán en dos escenarios de simulación distintos, la tarea 1 y 2 en el primer escenario, y la 3 y la 4 en el segundo.

La tarea 1 no se utiliza el robot, se trata de una tarea de carácter de percepción de las cámaras del entorno. Hay que identificar un gesto señalador del operario, así como reconocer el punto de la pieza que señala, y estimar la posición de la pieza. Todo esto utilizando las cámaras kinect.

La tarea 2 consiste en la programación de nuestro robot para que realice una secuencia de remachado.

Las tareas 3 y 4 consisten en control de coordinación de dos brazos manipuladores robóticos, en la tarea 3 se trata de introducir un objeto que porta un brazo en otro objeto que porta el otro, y la tarea 4 se trata de mover un objeto de forma coordinada entre ambos manipuladores esquivando un obstáculo del escenario.

Figura 3: EuRoC participantes.

2 ESTADO DEL ARTE DE LA ROBÓTICA

Antes de comenzar con el planteamiento de la tarea encomendada y su posterior desarrollo de la solución, daremos un breve repaso sobre la robótica en general, sus inicios, su historia… hasta centrarnos en lo relevante en nuestro proyecto, el robot industrial y la simulación robótica.

2.1 La Robótica

El estudio de la robótica requiere de un amplio conocimiento de la misma, y aunque es un área relativamente nueva, los avances realizados en sus pocos años de historia han sido muy importantes.

El término robótica procede de la palabra robot. El término robot fue introducido por el checo Karel Capek en 1921, y viene de la combinación de las palabras checas "*robota*", que significa "*trabajo obligatorio*" y "*robotnik*", que significa ciervo. La robótica es, por tanto, la ciencia o rama de la ciencia que se ocupa del estudio, desarrollo y aplicaciones de los robots.

Otra definición de robótica es el diseño, fabricación y utilización de máquinas automáticas programables con el fin de realizar tareas repetitivas como el ensamble de automóviles, aparatos, etc. y otras actividades. Básicamente, la robótica se ocupa de todo lo concerniente a los robots, lo cual incluye el control de motores, mecanismos automáticos neumáticos, sensores, sistemas de cómputos, etc.

La robótica [2] es una disciplina con sus propios problemas, sus fundamentos y sus leyes, Tiene dos vertientes: teórica y práctica. En el aspecto teórico de aúnan las aportaciones de la automática, informática e inteligencia artificial. Por el lado práctico o tecnológico hay aspectos de construcción (mecánica, electrónica), y de gestión (control, programación). La robótica presenta por lo tanto un marcado carácter interdisciplinario.

En la robótica se aúnan para un mismo fin varias disciplinas afines, pero diferentes, como la Mecánica, la Electrónica, la Automática, la Informática, etc. Los tres principios o leyes de la robótica según Asimov son:

- Un robot no puede dañar ni permitir que sea dañado ningún ser humano.
- El robot debe obedecer a todas las órdenes de los humanos, excepto las que contraigan la primera ley.
- El robot debe autoprotegerse, salvo que para hacerlo entre en conflicto con la primera o segunda ley.

Los robots son dispositivos compuestos de sensores que reciben datos de entrada y que pueden estar conectados a la computadora. Esta, al recibir la información de entrada, ordena al robot que efectúe una determinada acción. Puede ser que los propios robots dispongan de microprocesadores que reciben el input de los sensores y que estos microprocesadores ordenen al robot la ejecución de las acciones para las cuales está concebido. En este último caso, el propio robot es a su vez una computadora.

2.2 Historia de la Robótica

Por siglos, el ser humano ha construido máquinas que imitan partes del cuerpo humano. Los antiguos egipcios unieron brazos mecánicos a las estatuas de sus dioses; los griegos construyeron estatuas que operaban con sistemas hidráulicos, los cuales eran utilizados para fascinar a los adoradores de los templos.

El inicio de la robótica actual puede fijarse en la industria textil del siglo XVIII, cuando Joseph Jacquard inventa en 1801 una máquina textil programable mediante tarjetas perforadas. Luego, la Revolución Industrial impulsó el desarrollo de estos agentes mecánicos. Además de esto, durante los siglos XVII y XVIII en Europa fueron construidos muñecos mecánicos muy ingeniosos que tenían algunas características de robots. En 1805, Henri Maillardert construyó una muñeca mecánica que era capaz de hacer dibujos.

La palabra robot, como bien se desarrolló antes comenzó a utilizarse en 1921 en la obra del dramaturgo checo Karel Capek "*Los Robots Universales de Rossum*", donde un hombre fabricó un robot que luego éste mataría a

su creador.

Son varios factores que intervienen para que se desarrollaran los primeros robots en la década de los 50's. La investigación en inteligencia artificial desarrolló maneras de emular el procesamiento de información humana con computadoras electrónicas e inventó una variedad de mecanismos para probar sus teorías. Las primeras patentes aparecieron en 1946 con los muy primitivos robots para traslado de maquinaria de Devol. En 1954, con la aparición de las primeras computadoras, Devol diseña el primer robot programable

En los 60's se instaló en la Ford Motors Company el robot Unimate, basado en la transferencia de artículos. Ya en los 70's, la Standford University desarrolló un pequeño brazo robótico de accionamiento eléctrico, bautizado como "Standford Arm".

Actualmente, el concepto de robótica ha evolucionado hasta los sistemas móviles autónomos, que son aquellos que son capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión.

En los setenta, la NASA inició un programa de cooperación con el Jet Propulsión Laboratory para desarrollar plataformas capaces de explorar terrenos hostiles.

En la actualidad, la robótica se debate entre modelos sumamente ambiciosos, como es el caso del COG "robot de cuatro sentidos", desarrollado en el Instituto Tecnológico de Massachusetts (MIT), con proceso de aprendizaje cognitivo propio, vehículos con control remoto como SOUJOURNER O LUNA ROVER, o los robots mascotas de Sony.

En general la historia de la robótica la podemos clasificar en cinco generaciones: las dos primeras, ya alcanzadas en los ochenta, incluían la gestión de tareas repetitivas con autonomía muy limitada. La tercera generación incluiría visión artificial, en lo cual se ha avanzado mucho en los ochenta y noventas. La cuarta incluye movilidad avanzada en exteriores e interiores y la quinta entraría en el dominio de la inteligencia artificial en lo cual se está trabajando actualmente.

2.3 Clasificación de Robots

De manera general, y basándose en su morfología, los robots se suelen dividiren los siguientes tipos:

- **Robot industrial o manipulador**

Son artilugios mecánicos y electrónicos destinados a realizar de forma automática determinados procesos de fabricación o manipulación. Se utilizan principalmente en la fabricación industrial.

Los robots industriales son, con diferencia, el tipo de robot más utilizado, siendo Estados Unidos y Japón los líderes tanto en su fabricación como en su consumo.

Figura 4: Robot industrial o manipulador.

- **Robots móviles**

Los robots móviles están provistos de algún tipo de mecanismo que les permite desplazarse de lugar autónomamente, como pueden ser patas, ruedas u orugas y reciben la información del entorno con sus propios sistemas sensores. Son empleados en plantas industriales para el transporte de mercancías y para la exploración de lugares de difícil acceso o muy distantes, como es el caso dela exploración espacial y de las investigaciones o rescates submarinos.

- **Androides o humanoides**

Intentan reproducir total o parcialmente la forma y el comportamiento del ser humano. Actualmente, los androides están bastante evolucionados, sobre todo en Japón, pero aún no tienen una utilidad práctica por sus propias limitaciones y por su precio de fabricación. Básicamente están destinados a la investigación y a tareas de marketing de las propias empresas desarrolladoras.

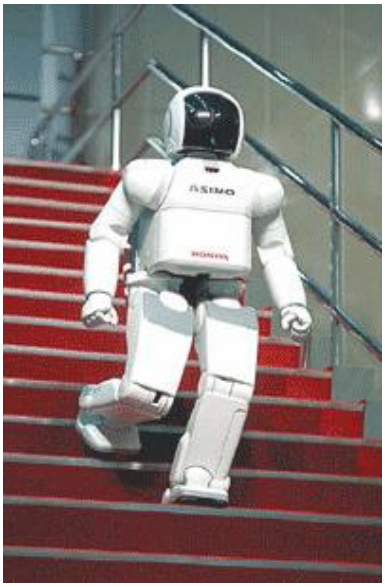

Figura 5: Robot humanoide.

- **Zoomórficos**

Los robots zoomórficos reproducen con mayor o menor grado de realismo, los sistemas de locomoción de diversos seres vivos.

A continuación, detallaremos y nos centraremos en el robot usado en el proyecto, el robot manipulador o industrial.

2.4 Robot manipulador o industrial

Los robots manipuladores o robots industriales (conocidos así porque inicialmente fueron usados masivamente en la industria) fueron los encargados de inaugurar la era de los robots en los años 60, con la herencia adquirida de los primeros teleoperadores. Por ello, es el área de la robótica donde la investigación está más avanzada.

Es difícil encontrar investigaciones que trabajen en los aspectos básicos de los robots manipuladores tradicionales y, las pocas que hay, están centradas en la inclusión de modernos sensores o actuadores. Un ejemplo de esto es la adición de cámaras para reconocimiento avanzado de imágenes que mejoren la efectividad de dichos robots.

El área más interesante, y donde sí que se investiga de manera importante es en la búsqueda de novedosas aplicaciones para los robots manipuladores, como es el caso de los robots quirúrgicos, los cuales están teniendo un gran auge en estos momentos. En este caso, la investigación no está centrada en el robot sino en adaptar éste a las necesidades propias de la aplicación.

2.4.1 Estructura mecánica

Se conoce también al robot industrial como *brazo robótico* y esto se debe a que guarda cierta similitud con la anatomía del brazo humano. También por ello, para hacer referencia a los distintos elementos que componen el robot, se utilizan términos como cuerpo, brazo, codo o muñeca.

Formalmente, se denomina *base* al punto de apoyo del robot, generalmente sujeto de forma fija al suelo. Los elementos o *eslabones* van unidos por medio de diferentes *articulaciones*, que permiten un movimiento relativo entre dos eslabones consecutivos. En la parte final, se sitúa el *efector final*, que son los encargados de interaccionar directamente con el entorno del robot.

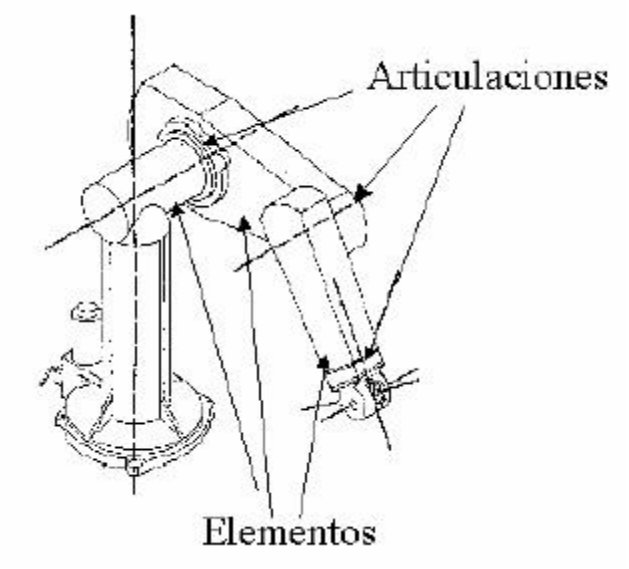

Figura 6: Estructura mecánica del manipulador.

El movimiento de cada articulación puede ser de desplazamiento, de giro o de una combinación de ambos. De este modo son posibles los 5 tipos diferentes de articulaciones, con sus diferentes grados de libertad, que pueden ser articulación de **rotación**(1), **prismática**(1), **cilíndrica**(2), **planar**(2) y **esférica**(3).

Los ***grados de libertad*** son el número de movimientos independientes que puede realizar cada articulación con respecto a la anterior. El número de grados de libertad de un robot viene dado por la suma de los grados de libertad de cada una de sus articulaciones.

El empleo de diferentes combinaciones de articulaciones en un robot, da lugar a diferentes configuraciones con características a tener en cuenta tanto en el diseño y construcción del robot como en su aplicación.

2.4.2 Aplicaciones

Los robots manipuladores tienen su principal foco de trabajo en la industria, automatizando los procesos de producción o almacenaje. Generalmente no trabajan de forma independiente sino en conjunto con otras máquinas y herramientas formando células de trabajo. Se enumeran a continuación algunos ejemplos:

- Operaciones de procesamiento, como soldadura, pintura, etc. Este tipo de robots son muy comunes en la industria de la automoción.
- Operaciones de ensamblaje, donde el trabajo repetitivo facilita el uso de este tipo de robots.
- Operaciones de empaque (en tarimas o pallets), agilizando el proceso y manejando grandes pesos.
- Otro tipo de operaciones como pueden ser remachados, estampados, corte por choro de agua, sistemas de medición, etc.

Concretamente, el robot manipulador del proyecto tiene una aplicación industrial de proceso de remachado a una pieza de trabajo provista de agujeros.

2.4.2.1 Remachados mediante Robot

Con el fin de conseguir eficiencia y flexibilidad en la industria, se han desarrollado robots destinados a realizar tareas como remachado o atornillado, liberando así una gran carga de trabajo para el operario, por no hablar de la mayor efectividad conseguida con el robot.

Los robots de remachado son aquellos utilizados para sellar materiales o artículos metálicos. Generalmente se utlizan en informática y electrónica, así como en la fabricación y ensamblado de electrodomésticos. Estos robots realizan una misma tarea con gran efectividad, como por ejemplo también podría ser el atornillado. Estos remaches se pueden realizar también mediante soldadura, en función de lo que se programe en cada caso y según las necesidades del cliente.

En el mercado actual podemos encontrar numerosas empresas dedicadas a la comercialización de robots y sus fabricantes como pueden ser Stäubli, Yaskawa o Adept Technology destinadas a distintos sectores como la carpintería metálica, o los electrodomésticos, informática o electrónica, citados anteriormente.

Figura 7: Robot de remachado Stäubli TX90.

2.4.3 Nuevas estructuras de robots manipuladores

El robot manipulador tradicional que se ha presentado anteriormente es el más usado, pero desde hace tiempo se investiga mucho en novedosas estructuras para este tipo de robots:

- **Robots manipuladores redundantes**: Básicamente se componen de un gran número de eslabones, los cuales además tienen la cualidad de ser iguales y repetitivos. Se denominan también robots de tipo serpiente. Estos robots tienen la capacidad de introducirse por espacios poco estructurados debido a su flexibilidad.

- **Robots manipuladores antropomorfos** (generalmente imitando la forma de la mano humana): Su aplicación se aleja de la industria tradicional y se acerca más a la creación de prótesis para la investigación médica.

2.5 La simulación Robótica

La simulación juega un papel crucial previo en los proyectos robóticos gracias a que nos permite realizar las operaciones de validación y verificación de aplicaciones robóticas antes de la construcción física del robot, lo que conlleva un importante ahorro tanto de tiempo como de dinero. Su empleo no sólo se basa en un carácter, por llamarlo de alguna manera, preventivo del producto. La simulación virtual también tiene una importante aplicación a la hora de entrenar el manejo y comprender las características del dispositivo antes de emplear el robot real.

Simuladores basados en comportamientos robóticos nos permiten crear mundos simplificados con objetos rígidos y programar robots que interactúen con ellos. En ocasiones, un entorno con condiciones extremas u operaciones en un área remota no nos permiten verificar todos los procedimientos y problemas físico-mecánicos ni por ello testear de una forma sólo experimental. Por lo tanto, pueden presentarse ocasiones en las que haya que tomar decisiones con el único análisis de los resultados de una simulación. Una de las aplicaciones más populares es el modelado 3D y su representación que requieren de buenos motores de leyes físicas y buenos gráficos para una emulación aceptable del robot y el mundo que lo rodea. Esto implica que cada robot tendrá unas propiedades gráficas y unas físicas.

Las técnicas de simulación e implementación robótica necesariamente pasan por una descripción analítica del sistema físico-mecánico. Sin embargo, a veces puede no contarse con todos los parámetros reales para simular un entorno cien por cien realista. Se emplean entonces técnicas de manejo de datos con la probabilidad y estadística. Es necesario también el modelado numérico con su respectivo software y toolboxes. Son empleados simuladores multidominio e híbridos con métodos que soporten una simulación rápida en tiempo real.

En la actualidad, la simulación robótica nos proporciona una grandísima variedad de elementos: diferentes familias de robots (UGV para tierra, UAV para aire, AUV para agua, brazos robóticos, manos robóticas, humanoides, avatares…), actuadores (Cadenas cinemáticas genéricas, actuadores de fuerza controlada…), subactuadores, sensores (Odometría, IMU, GPS, cámaras, lásers, emisores y receptores…), mecanismos, manipuladores, herramientas robóticas…). Llegando incluso a existir simuladores empleados en tareas en el espacio como satélites o aeronaves.

Científicos e ingenieros han venido desarrollando conjuntamente una gran variedad de técnicas de modelado y simulación creando diferentes softwares, de los cuales destacaré a continuación los más importantes. **Morse** es un simulador de un robot de código abierto con soporte completo a ROS . Usa OpenGL como su motor 3D y tienen un render realista, que hace que la simulación sea más sencilla de entender. El sistema funciona con comandos en línea, pero también puede usarse Python para controlar los robots en el simulador. Un componente sorprendente de Morse es su habilidad para modelar la interacción humano/robot. Otro simulador similar es **Gazebo**, también para ROS. Como es el usado en este trabajo, lo desarrollaré más adelante. El siguiente ejemplo es **Webots**, el cual utiliza ODE (Open Dynamics Engine) para la detección de colisiones, proporcionando también una simulación precisa de velocidad, inercia y fricción. Una de las características que han hecho famoso **ARS** es que funciona exclusivamente con Python y su facilidad para generar la documentación. Otro simulador muy famoso es **V-Rep** no sólo porque los controladores pueden ser escritos en casi todos los lenguajes de programación existentes, sino por la versatilidad que le proporciona su distribuida arquitectura de control: cada objeto puede ser controlado independientemente por un script interno, un plugin, un nodo de ROS, un cliente remoto API o el cliente. Es empleado para desarrollo rápido de algoritmo y simulaciones de cadenas de montaje.

3 MODELADO, CONTROL Y SIMULACIÓN DE ROBOTS

En el campo de la robótica, para el modelado, simulación y control de robots es necesario disponer de unas herramientas que nos permitan conocer su posición, orientación, velocidad, configuración entre otros en el entorno de trabajo. Para ello es necesario conocer previamente como representar la posición y la orientación en el espacio, establecer un modelo tanto cinemático como dinámico que nos permita relacionar valores articulares-posición, velocidades cartesiana-velocidades articulares, par aplicado-articulación, etc.

Pueden encontrar esta información de forma más extensa en los libros de robótica generales:

- Ollero Baturone, (2007). *Robótica: Manipuladores y robots móviles.*
- Lewis, Frank L., Abdallah, C. T. Dawson, D. M. (1993). *Control of robot manipulators.*

3.1 Representación de posición y orientación

Sabemos que la posición de un punto en el espacio euclídeo tridimensional viene determinada por tres cantidades, que llamamos sus coordenadas, y decimos que están expresadas en algún sistema de referencia, formado por tres ejes, usualmente rectilíneos.

En lo sucesivo usaremos exclusivamente sistemas de referencia rectilíneos, ortogonales (es decir, con sus tres ejes perpendiculares dos a dos), normalizados (es decir, las longitudes de los vectores básicos de cada eje son iguales) y dextrógiros (el tercer eje es producto vectorial de los otros dos).

Usaremos, pues, simplemente el término "sistema" para referirnos a sistemas ortonormales.

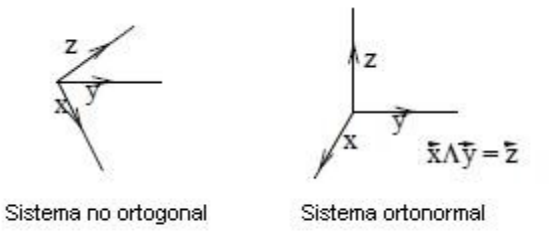

Figura 8: Tipos de sistemas de referencia.

Las coordenadas de un punto, denotadas por (x; y; z), son las proyecciones de dicho punto perpendicularmente a cada eje, o, equivalentemente, las componentes del vector que lo une al origen de coordenadas. En lugar de usar estas, nos será más conveniente el uso de las llamadas coordenadas homogéneas, en la forma:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w \end{pmatrix}$$ donde

$$\begin{aligned} x' &= xw \\ y' &= yw \\ z' &= zw \end{aligned}$$

siendo w una cantidad arbitraria, que se suele tomar como 1. Si, como resultado de algún cálculo, w fuese

distinto de 1, las coordenadas usuales se reconstruyen simplemente dividiendo las tres primeras coordenadas homogéneas entre esta cuarta.

La traslación de un punto x, y, z por un vector v es un punto x', y', z' tal que:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Pero también como el producto de una matriz por un vector homogéneo, en la forma:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Esto tiene la ventaja de que, si: $\vec{x}' = H \cdot \vec{x}$ $\vec{x} = (H)^{-1} \cdot \vec{x}'$

Donde se puede calcular la inversa, que resulta ser:

$$(H)^{-1} = \begin{pmatrix} 1 & 0 & 0 & -v_x \\ 0 & 1 & 0 & -v_y \\ 0 & 0 & 1 & -v_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

lo cual es consistente con el hecho de que x esta trasladado por un vector –v respecto a x'.

Respecto a la rotación alrededor de un eje, en el caso bidimensional, se está rotando con respecto a un eje z perpendicular al plano de la figura.

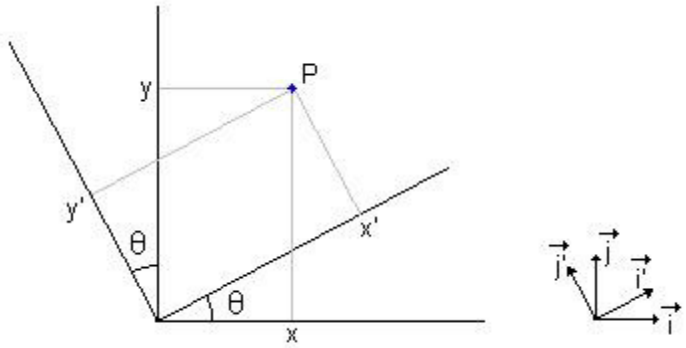

Figura 9: Rotación de sistema de referencia respecto al eje Z

Llamando i, j a los vectores básicos del sistema original, e i' y j' a los del sistema girado, se tiene que

$$\vec{x} = x\vec{i} + y\vec{j} = x'\vec{i}' + y'\vec{j}'$$

$$\begin{aligned} \vec{i}' &= \cos\theta\vec{i} + \sin\theta\vec{j} \\ \vec{j}' &= -\sin\theta\vec{i} + \cos\theta\vec{j} \end{aligned}$$

$$x\vec{i} + y\vec{j} = x'(\cos\theta\vec{i} + \sin\theta\vec{j}) + y'(-\sin\theta\vec{i} + \cos\theta\vec{j})$$

Es decir, que

Igualando componente a componente, escribimos la matriz como

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

Si generalizamos a tres dimensiones, como la coordenada z no varia y la cuarta coordenada homogénea sigue siendo 1, tenemos

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

Para hallar la transformación inversa basta ver que desde el punto de vista de R', R esta rotado un ángulo -θ , luego podemos afirmar que

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) & 0 & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

Esta operación es más simple que invertir la matriz, aunque por supuesto, equivalente.

En general, si hubiéramos rotado alrededor de otro de los ejes básicos, se puede ver que

$$Rot(x,\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Rot(y,\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Rot(z,\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cabe destacar el cambio de signo en la rotación alrededor del eje y, debido a que, si el eje alrededor del cual rotamos nos apunta, los otros dos forman un ángulo de 90o en el caso de x y z, pero de -90o en el caso de y. Se pueden aplicar a un punto tantas transformaciones sucesivas (rotaciones y traslaciones) como se quiera. La operación resultante vendría dada por una matriz que sería producto de las matrices de cada operación, aplicadas en el orden correcto, dado que el producto de matrices no es conmutativo.

Se pone más a la derecha la primera transformación que se aplique, siendo expresado como

$$Y = T_2 R_3 T_1 R_2 R_1 X$$

Significa que se aplica al punto X la rotación 1, seguida de la rotación 2, seguida de la traslación 1, luego la rotación 3 y por último la traslación 2.

Veamos ahora cual sería la matriz de rotación respecto a un eje cualquiera. Sea un eje que pasa por el origen definido por un vector unitario alrededor del cual giraremos un ángulo θ.

Esta rotación se podrá descomponer en tres rotaciones sobre los ejes básicos, lo que equivaldrá a:

– Rotar un ángulo α alrededor de x, con lo que P pasara a la posición P'.

– Rotar un ángulo -β alrededor de y, con lo que P' pasara a la posición P".

– Rotar un ángulo θ alrededor de z, que es la rotación que se pide.

– Rotar un ángulo β alrededor de y, deshaciendo la segunda rotación

– Rotar un ángulo -α alrededor de x, deshaciendo la primera rotación.

Entonces tenemos que $R_{\vec{r},\theta} = R_{x,-\alpha} R_{y,\beta} R_{z,\theta} R_{y,-\beta} R_{x,\alpha}$

O también

$$R_{\vec{r},\theta} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha & s\alpha & 0 \\ 0 & -s\alpha & c\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c\beta & 0 & s\beta & 0 \\ 0 & 1 & 0 & 0 \\ -s\beta & 0 & c\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c\theta & -s\theta & 0 & 0 \\ s\theta & c\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot$$

$$\cdot \begin{pmatrix} c\beta & 0 & -s\beta & 0 \\ 0 & 1 & 0 & 0 \\ s\beta & 0 & c\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha & -s\alpha & 0 \\ 0 & s\alpha & c\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Debemos destacar que cualquier secuencia consecutiva de transformaciones se puede especificar de dos formas:

– Realizando la rotación que lleva un sistema al otro, alrededor de los ejes iniciales.

– Realizando la rotación que lleva un sistema al otro alrededor de uno de los ejes girados, es decir, los que resultaron de la última transformación.

En el primer caso, la matriz que describe esta transformación deberá pre-multiplicarse por la que describía las transformaciones efectuadas hasta el momento, obteniendo la transformación total.

En el segundo caso la matriz que describe esta transformación deberá postmultiplicarse por la que describía las transformaciones efectuadas hasta el momento, obteniendo la transformación total.

3.2 Modelo cinemático del robot manipulador

Estudiando los modelos geométricos y cinemáticos de los robots podemos establecer la relación entre variables asociadas a las articulaciones del robot y su situación en un sistema de referencia, es decir, en función de sus valores articulares conocer su posición (cinemática directa) y viceversa (cinemática inversa).

3.2.1 Cinemática directa

El problema cinemático directo se plantea en términos de encontrar una matriz de transformación que relaciona el sistema de coordenadas ligado al cuerpo en movimiento respecto a un sistema de coordenadas que se toma como referencia. Para lograr esta representación se usan las matrices de transformación homogénea 4x4, la cual incluye las operaciones de traslación y la orientación. La matriz de transformación homogénea es una matriz de 4x4 que transforma un vector expresado en coordenadas homogéneas desde un sistema de coordenadas hasta otro sistema de coordenadas.

La matriz de transformación homogénea tiene la siguiente estructura:

$$T = \begin{bmatrix} \text{matriz de rotación} & \text{vector de posición} \\ f_{1x3} & \text{escalado} \end{bmatrix} = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} n & s & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

donde los vectores n, s, a, son vectores ortogonales unitarios y p es un vector que describe la posición x, y, z del origen del sistema actual respecto del sistema de referencia.

Para entender las propiedades de la matriz de transformación homogénea nos fijamos en la siguiente figura.

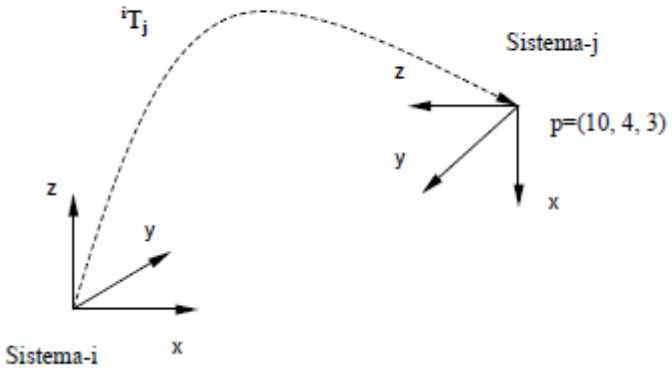

Figura 10:Interpretación geométrica de la matriz de transformación homogénea.

$${}^{i}T_{j} = \begin{bmatrix} 0 & 0 & -1 & 10 \\ 0 & -1 & 0 & 4 \\ -1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Al analizar las columnas de la submatriz de rotación de la matriz de transformación homogénea ${}^{i}T_{j}$, un observador localizado en el origen de sistema-i, puede ver cómo están orientados los ejes x, y, z del sistema-j, además también observa como se ha desplazado en coordenadas cartesianas el origen del sistema-j respecto del origen del sistema de referencia con la información del vector de posición.

3.2.1.1 Representación Denavit-Hartenberg

La representación de D-H, se aplica a robots de cadena cinemática abierta y consiste en una serie de reglas para colocar los sistemas de referencia de cada eslabón del robot.

Antes de aplicar el método de D-H es importante tener en cuenta los siguientes comentarios:

- Se parte de una configuración cualesquiera del robot, si bien es aconsejable colocarlo en una posición sencilla de analizar.
- Se numeran los eslabones, asignando el *0* para la base y *n-1* para el último eslabón, siendo *n* el número de grados de libertad del robot.
- El sistema de coordenadas ortonormal dextrógiro de la base (x0, y0, z0) se establece con el eje z0 localizado a lo largo del eje de movimiento de la articulación 1 y apuntando hacia fuera del hombro del brazo del robot.

- El sistema de referencia de cada eslabón se coloca al final del mismo, en el extremo de la articulación a la cual está conectado el eslabón siguiente.
- El ángulo ó desplazamiento de cada eslabón siempre se mide tomando como base el sistema de referencia del eslabón anterior.
- Al colocar el sistema de referencia del eslabón-i, se deben seguir las siguientes reglas:
  - El eje z_i del sistema de referencia debe quedar alineado a lo largo de la articulación
  - El eje x_i debe colocarse con orientación normal al plano formado por los ejes z_{i-1} y z_i
- Al establecer los sistemas de coordenadas de la mano se debe tener en cuenta el principio de Pieper's en el cual se establece que los tres últimos sistemas de referencia se intercepten en un punto, lo cual permite obtener una solución para el problema cinemático inverso de forma cerrada para estas articulaciones.

Cada sistema de coordenadas se establece sobre las siguientes reglas.

- θ_i: Es el ángulo de la articulación desde el eje x_{i-1} hasta el eje x_i, medido respecto del eje z_{i-1}, usando la regla de la mano derecha.
- d_i: Es la distancia medida desde el origen del sistema i-1, a lo largo del eje z_{i-1} hasta la intersección del eje z_{i-1} con el eje x_i.
- a_i: Es la distancia de separación entre los orígenes de los sistemas de referencia i-1 e i, medida a lo largo del eje xi hasta la intersección con el eje z_{i-1}. (o la distancia más corta entre los ejes zi-1 y zi, cuando estos no se interceptan).
- α_i: Es el ángulo que separa los ejes z_i y z_{i-1}, medido respecto del eje x_i.

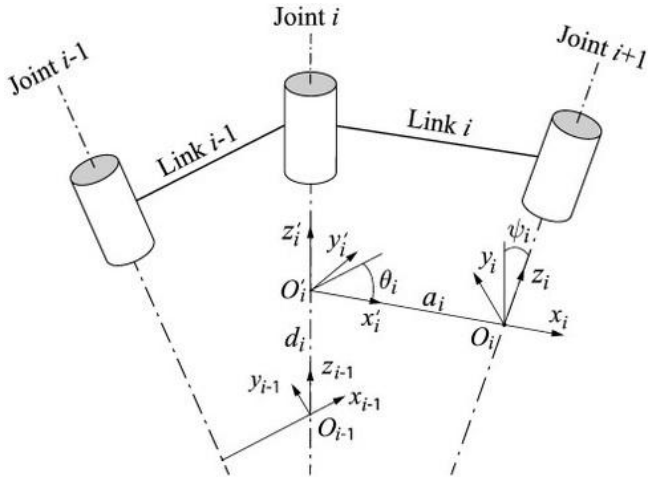

Figura 11: Parámetros D-H.

De acuerdo a las reglas de D-H, se determina la siguiente matriz de transformación homogénea:

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \mathbf{d}_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{c}\theta_i & -\mathbf{s}\theta_i & 0 & 0 \\ \mathbf{s}\theta_i & \mathbf{c}\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \mathbf{a}_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \mathbf{c}\alpha_i & -\mathbf{s}\alpha_i & 0 \\ 0 & \mathbf{s}\alpha_i & \mathbf{c}\alpha_i & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} $$

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} \mathbf{c}\theta_i & -\mathbf{c}\alpha_i\mathbf{s}\theta_i & \mathbf{s}\alpha_i\mathbf{s}\theta_i & \mathbf{a}_i\mathbf{c}\theta_i \\ \mathbf{s}\theta_i & \mathbf{c}\alpha_i\mathbf{c}\theta_i & -\mathbf{s}\alpha_i\mathbf{c}\theta_i & \mathbf{a}_i\mathbf{s}\theta_i \\ 0 & \mathbf{s}\alpha_i & \mathbf{c}\alpha_i & \mathbf{d}_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

3.2.2 Cinemática inversa

La cinemática inversa consiste en hallar los valores de las coordenadas articulares del robot $q=[q1, q2, \ldots, qn]^T$ conocida la posición y orientación del extremo del robot.

Se pueden encontrar diversos métodos genéricos para la resolución de la cinemática inversa que pueden ser implementados en computadora, suele ser habitual la resolución por medio de métodos geométricos. La mayor parte de los robots suelen tener cadenas cinemáticas relativamente sencillas, que facilitan la utilización de los métodos geométricos. Para muchos robots, si se consideran sólo los tres primeros grados de libertad, se tiene una estructura planar. Este hecho facilita la resolución del problema. Asimismo los últimos tres grados de libertad suelen usarse para la orientación de la herramienta, lo cual permite una resolución geométrica desacoplada de la posición de la muñeca del robot y de la orientación de la herramienta.

3.3 Velocidades: Jacobiano de un robot manipulador

En esta sección se presenta la cinemática diferencial, la cual consiste en relacionar las velocidades angular y lineal del efector final con las velocidades articulares.

Matemáticamente, las ecuaciones de la cinemática directa definen una función entre el espacio cartesiano de posiciones y orientaciones y el espacio de las posiciones de las articulaciones. Las relaciones de velocidad están definidas por el Jacobiano de esta función. El Jacobiano o matriz Jacobiana J constituye una de las herramientas más importantes para la caracterización del robot. De hecho, es útil para encontrar configuraciones singulares, análisis de redundancia, el mapeo entre las fuerzas aplicadas por el efector final y torques resultantes en las articulaciones, así como la obtención de las ecuaciones de movimiento del manipulador.

$$\dot{p} = \frac{\partial p}{\partial q}\dot{q} = \boldsymbol{J}_p(q)\dot{q}. \qquad \dot{\phi} = \frac{\partial \phi}{\partial q}\dot{q} = \boldsymbol{J}_\phi(q)\dot{q}.$$

Existen dos tipos de Jacobianos: Jacobiano geométrico, cuyo mapeo, representado por una matriz, depende de la configuración del robot, y, alternativamente, el Jacobiano analítico, el cual es posible calcular vía la diferenciación de la función de cinemática directa con respecto a las variables articulares, siempre y cuando la localización del efector final sea expresada con referencia a una representación mínima en el espacio operacional. El Jacobiano analítico, en general, difiere del geométrico, ya que éste es adoptado cuando cantidades del espacio operacional son de interés y el jacobiano geométrico es adoptado cuando cantidades físicas son de interés.

3.4 Modelo dinámico de un robot manipulador

El modelo de un robot manipulador consiste en encontrar la relación entre las fuerzas ejercidas sobre la estructura y las posiciones, velocidades y aceleraciones de las articulaciones. Para un manipulador de n grados de libertad y articulaciones de revolución, la energía cinética se puede calcular como:

$$K = \frac{1}{2}\dot{q}^T \boldsymbol{H}(q)\dot{q},$$

Donde q es el vector e coordenadas generalizadas de las articulaciones y H(q) es la matriz de inercia simétrica positiva definida. Adicionalmente, la fricción viscosa se puede introducir tomando en consideración la función de disipación de Rayleigh.

$$D = \frac{1}{2}\dot{q}^T \boldsymbol{D}(q)\dot{q},$$

Donde D es una matriz diagonal semidefinida positiva que considera los coeficientes de fricción viscosa de las articulaciones. Sustituyendo estas ecuaciones, y tomando derivadas se obtiene la ecuación de movimiento

$$\boldsymbol{H}(q)\ddot{q} + \boldsymbol{C}(q,\dot{q})\dot{q} + \boldsymbol{D}\dot{q} + g(q) = \tau,$$

Donde τ es el vector de par actuando en las articulaciones y g(q) es el vector de pares gravitacionales.

3.5 Control de articulaciones de un robot manipulador

En este apartado se tratarán de forma breve los métodos para generar actuaciones que permiten controlar articulaciones de un robot manipulador. No entraremos en demasiados detalles ya que en la resolución de la tarea no realizamos ninguna estrategia de control, ya que para el movimiento de las articulaciones ya hay implementado un controlador por la organización EuRoC.

La estrategia más simple de control es el control de cada articulación de forma independiente o "desacoplada" del resto de las articulaciones, usando habitualmente un controlador PID digital en cada articulación. Los pares de acoplamiento se tratan como perturbaciones que es necesario anular mediante el control. Se presenta el ajuste más conveniente de las ganancias del controlador PID, ya que, dependiendo de la carga e incluso la posición dentro del espacio de trabajo, se necesitarían valores diferentes de dichas ganancias.

Otra estrategia si se quiere considerar de forma apropiada los acoplamientos entre las articulaciones, siendo necesario aquí el modelo dinámico del manipulador. La dificultad de estas técnicas de control es el conocimiento de los parámetros del modelo dinámico de nuestro robot, ya que a veces son difíciles de medir o estimar, o pueden variar debido a desgastes o modificaciones en las condiciones de trabajo. Así se presenta el control basado en par computado, y se trata de compensar los efectos gravitatorios, centrífugos y rozamientos generando un par apropiado.

Por último, como ya mencionamos anteriormente, nuestra ley de control sobre nuestro manipulador consiste en un controlador PID para cada articulación que actúa de forma independiente, y de cuyos valores del controlador no podemos llegar y modificar.

4 ROS Y HERRAMIENTAS DE DESARROLLO

En este capítulo vamos a presentar el software de simulación usado en nuestro proyecto, donde encontramos desarrollado nuestro entorno de simulación, y donde debemos implementar nuestra solución.

4.1 ROS

ROS (Robot Operating System) [3] es un framework, basado en Ubuntu, para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. ROS se desarrolló originalmente en 2007 bajo el nombre de *switchyard* por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford (STAIR). Desde 2008, el desarrollo continúa primordialmente en Willow Garage, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado.

ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad 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 (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'.

ROS tiene 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 en inglés *stacks*) que implementan la funcionalidades tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc.

ROS es software libre bajo términos de licencia BSD. Esta licencia permite libertad para uso comercial e investigador. Las contribuciones de los paquetes en *ros-pkg* están bajo una gran variedad de licencias diferentes.

Las áreas que incluye ROS son:

- Un nodo principal de coordinación.
- Publicación o subscripción de flujos de datos: imágenes, estéreo, láser, control, actuador, contacto, etc.
- Multiplexión de la información.
- Creación y destrucción de nodos.
- Los nodos están perfectamente distribuidos, permitiendo procesamiento distribuido en múltiples núcleos, multiprocesamiento, GPUs y clústeres.
- Login.
- Parámetros de servidor.
- Testeo de sistemas.

Las áreas que incluirán las aplicaciones de los paquetes de ROS son:

- Percepción
- Identificación de Objetos
- Segmentación y reconocimiento
- Reconocimiento facial
- Reconocimiento de gestos
- Seguimiento de objetos
- Egomoción
- Comprensión de movimiento
- Estructura de movimientos (SFM)
- Visión estéreo: percepción de profundidad mediante el uso de dos cámaras
- Movimientos
- Robots móviles
- Control
- Planificación
- Agarre de objetos

4.2 Diseño modular y distributivo

Ros fue diseñado para ser lo más distributivo y modular posible, de modo que los usuarios pueden utilizar ROS tanto como deseen. Su modularidad le permite seleccionar y elegir qué partes son útiles y qué partes prefiere implementar uno mismo.
La naturaliza distributiva de ROS también fomenta una gran comunidad de paquetes contribuidas por usuarios que añaden un gran valor a la parte superior del núcleo del sistema ROS. En el último recuento se contaron más de 3000 paquetes en el ecosistema ROS, y eso que solo son los paquetes que la gente ha hecho público. Estos paquetes varían en la fidelidad, cubriendo desde pruebas de conceptos de implementación de nuevos algoritmos hasta drivers de alta capacidad y calidad industrial. La comunidad de usuarios de ROS construye sobre la parte superior de una infraestructura común para proporcionar un punto de integración que ofrece acceso a controladores de hardware, robots genéricos, herramientas de desarrollo, bibliotecas externas útiles, y mucho más.

4.3 Historia

ROS es un largo proyecto con muchos antepasados y colaboradores. La necesidad de un marco de colaboración abierto fue requerido por muchas personas en la comunidad de investigación robótica, y muchos proyectos se han creado para alcanzar este objetivo.

Varios esfuerzos en la Universidad de Stanford a mitad de los años 2000 envolviendo integrativamente tanto STanford Al Robot (STAIR) como el programa Personal Robots (PR) crearon prototipos internos de sistemas de software flexibles y dinámicos destinado al uso robótico. En 2007, Willow Garage, una incubadora cercana de visionarios robóticos, aportó importantes recursos para ampliar estos conceptos mucho más allá y crear implementaciones bien probadas. Este esfuerzo se vió impulsado por un sinfín de investigadores que contribuyeron con su tiempo y experiencia para las ideas principales de ROS y sus paquetes de software fundamentales. En todo momento, el software fue desarrollado en abierto usando una licencia de código abierto (BSD open-source), y poco a poco se ha convertido en una plataforma ampliamente utilizada en la comunidad de investigación robótica.

Desde sus inicios, ROS se desarrolló en múltiples instituciones y para múltiples robots, incluyendo muchas de las instituciones que recibieron los robots PR2 de Willow Garage. A pesar de que habría sido mucho más fácil para todos los contribuyentes poner su código en los mismos servidores, en los últimos años, el modelo "federado" se ha convertido en una de las grandes fortalezas del ecosistema ROS. Cualquier grupo puede comenzar su propio repositorio de código ROS en sus propios servidores, y mantener la propiedad y control por completo, sin necesidad de permiso de nadie. Si deciden poner su repositorio a disposición del público, pueden recibir el reconocimiento y crédito que se merecen sus logros, y beneficiar se de la información técnica específica y mejoras como todos los proyectos de software de código abierto.

4.4 Comunidad activa y colaborativa

En los últimos años, ROS ha crecido para contar con una gran comunidad de usuarios en todo el mundo. Históricamente, la mayoría de los usuarios se encontraban en los laboratorios de investigación, para cada vez más estamos viendo una adopción en el sector comercial, en particular en la industria y servicios robóticos.

La comunidad ROS es muy activa. Según las últimas mediciones, la comunidad ROS cuenta con más de 1500 en la lista de ROS distributiva, más de 3300 usuarios en la wiki de documentación colaborativa y unos 5700 usuarios en foro de la página web. La wiki cuenta con más de 22000 páginas y más de 30 ediciones diarias. En el foro tienen lugar unas 13000 preguntas hechas hasta la fecha, con una tasa de respuesta del 70%.

ROS no solo ofrece por sí mismo un gran valor a la mayoría de los proyectos, sino que también representa una oportunidad para establecer contactos y colaborar con los expertos en robótica mundiales que forman parte de la comunidad de ROS. Una de las filosofías básicas de ROS es compartir el desarrollo de componentes comunes.

4.5 Conceptos básicos

Los conceptos fundamentales de la aplicación de ROS son **nodos**, **mensajes**, **topics** y **servicios**.

Los **nodos** son procesos ejecutables. ROS es diseñado para ser modular en una escala de grano fino: Un sistema típicamente se compone de muchos nodos. En este contexto, el término "nodo" es intercambiable por "módulo". El uso del término "nodo" surge de visualizaciones de ROS basados en sistemas tiempo de ejecución: cuando muchos nodos se están ejecutando, se realizan las comunicaciones "peer-to-peer", es decir, el enlace entre los distintos nodos son estas comunicaciones.

Los nodos se comunican entre sí mediante paso de mensajes. Un **mensaje** es una estructura de datos que pueden ser distintos tipos (enteros, punto flotantes, booleanos, etc) apoyados por tipos primitivos y constantes. Los mensajes pueden estar compuestos de otros mensajes, y matrices de otros mensajes, anidados de forma profunda.

En nodo envía un mensaje mediante su publicación en un "**topic**". Un **topic** puede interpretarse como un buzón de mensajes dónde llega siempre un mismo tipo o estructura de dato, de esa forma, si un nodo está interesado en un determinado tipo de dato, se suscribirá en el topic correspondiente. Puede haber varios publicados y suscriptores para un solo topic, como un nodo puede publicar y estar suscritos a varios topics a la vez.

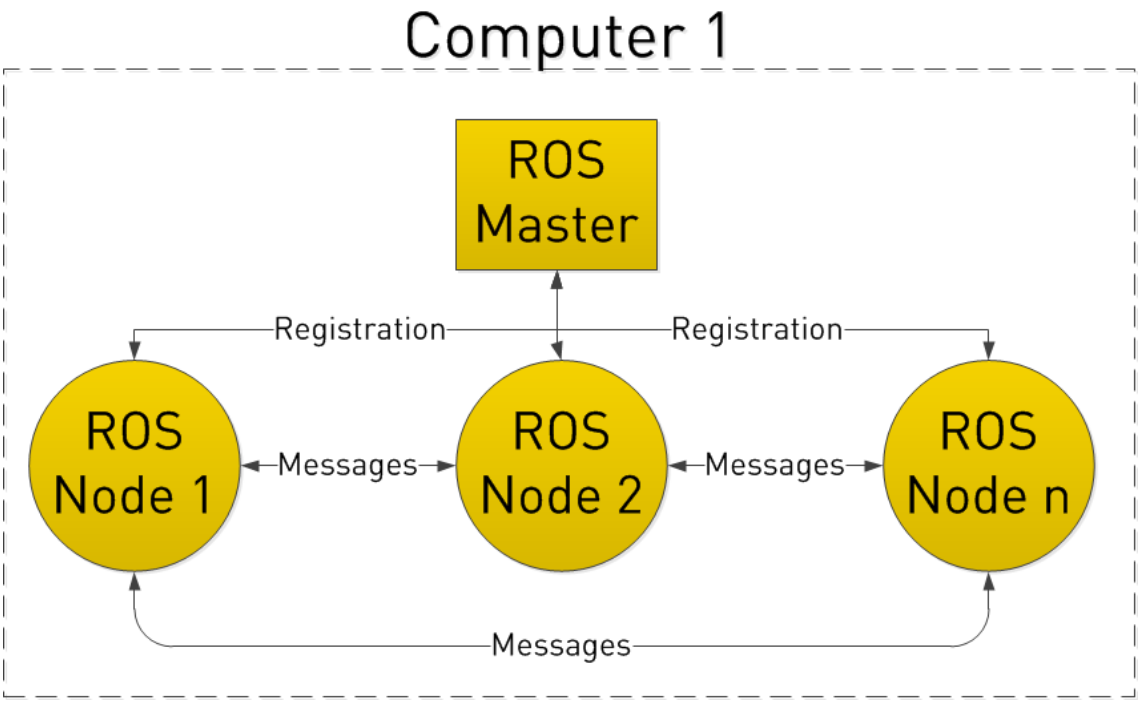

Figura 12: Sistema básico ROS.

Aunque el modelo de publicación-suscripción basado en topics es flexible, no es apropiado para transacciones síncronas, que pueden permitir el diseño de algunos nodos. Para esto, ROS nos facilita los "**servicios**", que se define con un nombre de cadena de caracteres y dos mensajes proporcionados: Uno de solicitud y otro de respuesta. Esto es análogo a los servicios web, que son definidos por los URI y tienen tipos de documentos de solicitud y respuesta bien definidos. A diferencia de los topics, solo un nodo puede requerir un servicio, mientras que será respondido por otro.

Finalmente también hay que destacar la capacidad de flexibilidad a la hora de la programación, ya que permite usar varios lenguajes de programación como C++, Python, Lisp u Octave, permitiendo además, que un sistema esté compuesto por nodos escritos en distinto lenguajes, aumentado su flexibilidad y modularidad.

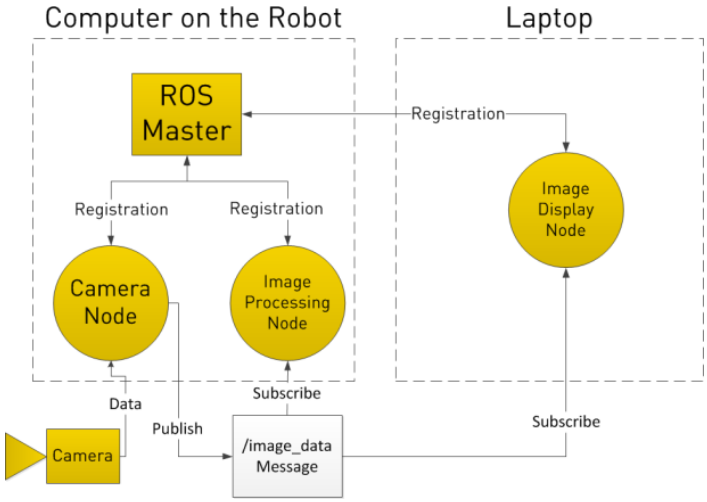

Figura 13: Sistema Robótico de visión en ROS.

4.6 Herramientas ROS

4.6.1 Gazebo

La simulación robótica es una herramienta esencial en el tool-box robótico. Un simulador bien diseñado permite probar rápidamente algoritmos, robots diseñados y realizar pruebas de regresión utilizando escenarios realistas. Gazebo [4] ofrece la posibilidad de simular con precisión y eficiencia poblaciones de robots en entornos interiores y exteriores complejos. Genera tanto la realimentación de los sensores como las interacciones físicas entre objetos tratándolos como cuerpos rígidos, a través de gráficos de alta calidad e interfaces programables. Además de tener una comunidad activa.

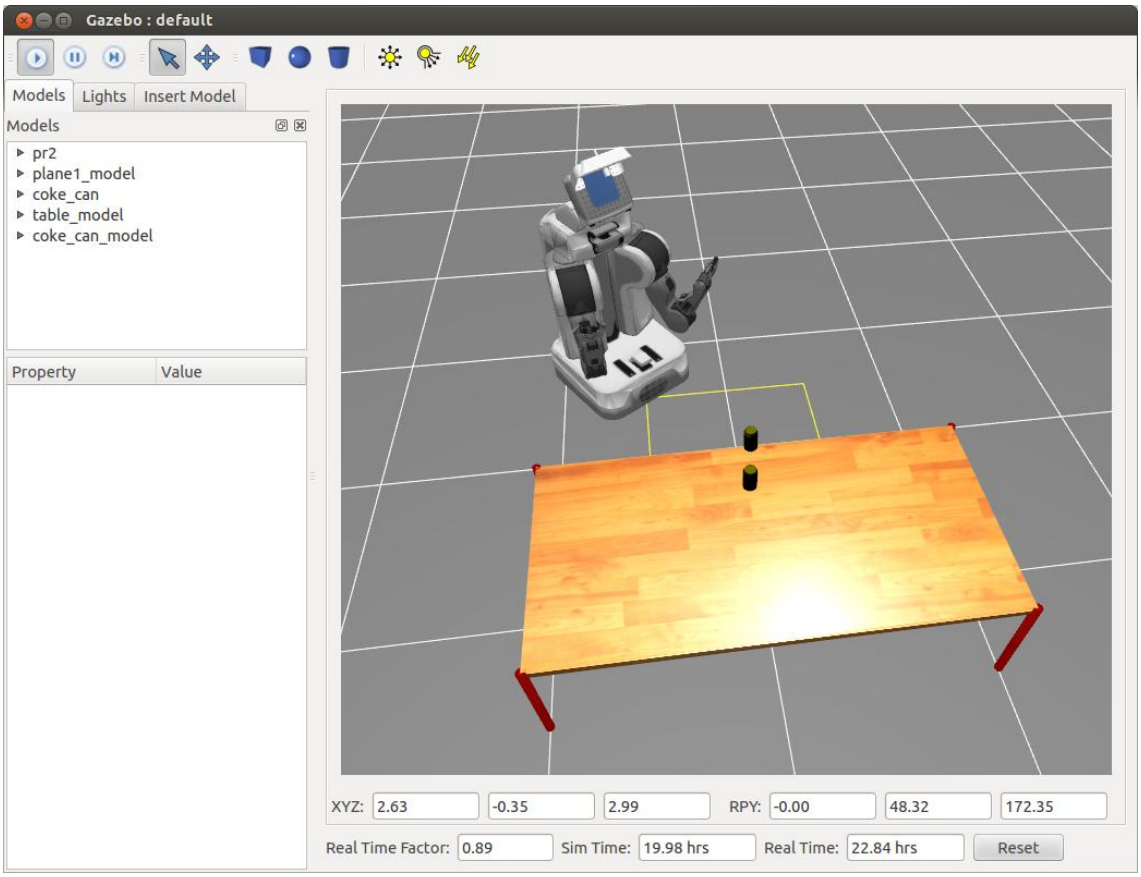

Figura 14: Gazebo.

4.6.2 RVIZ

RVIZ (ROS Visualization) [5] es un visualizador 3D que permite la visualización de datos de sensor e información de estado del sistema de ROS. Usando RVIZ, se puede ver la configuración actual de Baxter en un modelo virtual del robot, además de las representaciones en vivo de los valores de sensor publicados en los topics de ROS, incluyendo datos de cámara, mediciones de sensores de distancia infrarrojos, datos de sonar, etc. Algunos de los datos más importantes que podemos visualizar se encuentran:

- RobotModel: Muestra una representación visual de un robot en la posición dada (definida por la transformación TF del momento).
- TF: Descrita a continuación, nos permite visualizar los distintos ejes de referencia que conforma nuestro entorno.
- Point Cloud; Muestra los datos de una nube de puntos, configuradas por diferentes opciones disponibles.

- Camera: Crea una nueva ventana con la perspectiva de una cámara, y superpone la imagen en la parte superior de la misma.
- Laser Scan: Muestra los datos de una exploración por láser, con diferentes opciones para los modos de representación, acumulación, etc.
- Otras herramientas como Axes, Effort, Grid, Map, Markers, Path, Pose, Pose Array…etc.

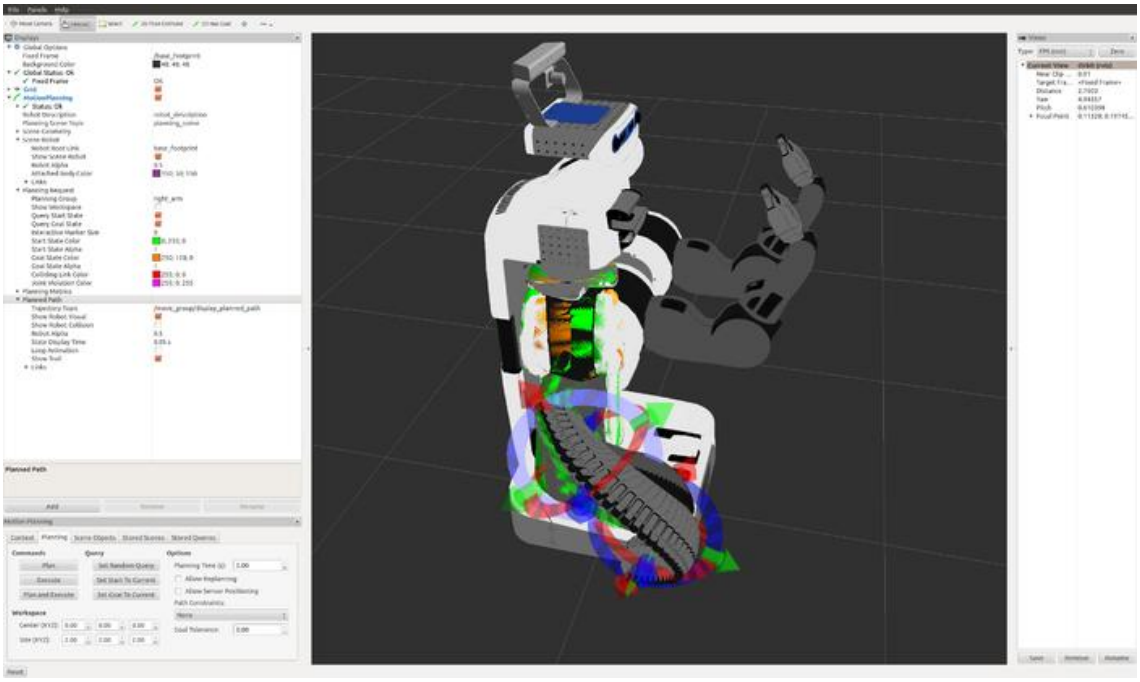

Figura 15: RVIZ.

4.6.3 TF

El sistema TF de ROS [6] permite coordinar múltiples sistemas de referencias y mantener la relación entre ellos en una estructura de árbol. TF se distribuye de manera que la información acerca de la coordinación de todos los sistemas de referencia del sistema está disponibles para todos los nodos de la red de ROS. Además del acceso a esta información, nos permite recuperar transformaciones entre ellos, transformar puntos, vectores y otras entidades entre dos ejes de referencia. Algunas aplicaciones son:

- tf_monitor: Imprime información sobre el actual árbol de coordenadas por consola.
- tf_echo: Imprime información sobre la transformación relativa entre dos ejes de referencia.
- static_transform_publisher: Publica un nuevo eje de referencia a través de una transformación estática de un eje ya existente.
- view_frames: Genera un PDF con nuestro árbol de tf.

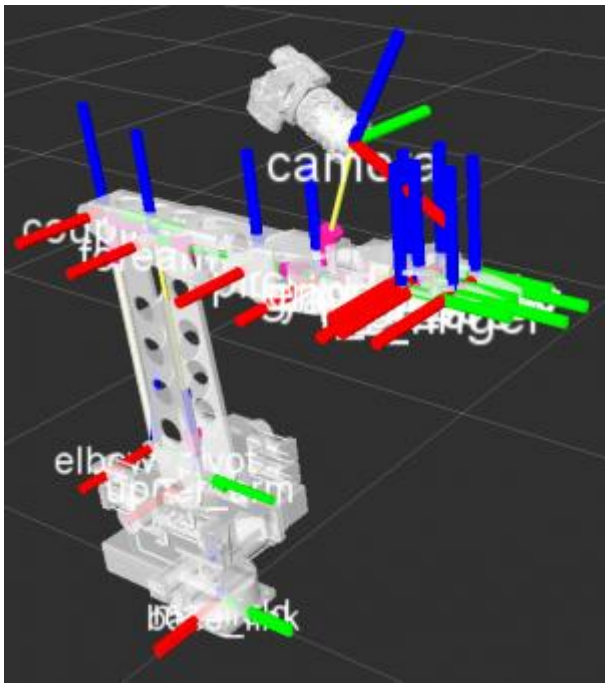

Figura 16: TF frames en RVIZ.

De aquí en adelante, a los ejes de referencia representados por TF los llamaremos "*TF frames"*.

Esta herramienta de ROS nos permitirá obtener posiciones relativas entre TF frames, que nos sirve por ejemplo, para calcular errores de posición entre ellos y que usaremos para estimar la posición del agujero deseado para remachar. También podremos realizar transformaciones entre estos marcos de referencia para las operaciones de cinemática inversa o directa del modelo cinemático, sin tener que usar matrices de transformación, como por ejemplo en el caso de aplicar la cinemática inversa, no usaremos el robot hasta el último eslabón como viene definido en el archivo URDF *tcp*, sino que tenemos que retroceder tres eslabones hacia atrás en el modelo cinemático hasta el eslabón *ee_link (end effector link)*.

4.6.4 URDF

URDF (Unified Robot Description Format) [7] se trata de una herramienta de ROS que consiste en un documento formato XML que nos permite representar el modelo de un robot.

Los archivos URDF se estructuran básicamente en forma de árbol, con dos componentes principales para construir el robot: Links (eslabones) y Joints (articulaciones):

- Links:

Constituyen las componentes físicas del robot, es decir, cada uno de los eslabones por lo que está formada nuestra cadena cinemática, también compone la parte visual donde debemos introducir nuestros archivos .DAE para su visualización posterior.

En el apartado visual -> geometry -> mesh podemos introducir nuestro modelo .dae diseñado previamente. También podemos introducir de forma genérica objetos geométricos simples como cilindros, esferas, cubos, etc… definiendo todos sus parámetros (arista, radio…).

- Joints:

Mediante la definición de los "joints" establecemos la relación entre los distintos links o eslabones que componen nuestro robot, tanto las posiciones relativas entre ellos, como la articulación que habrá entre ellos.

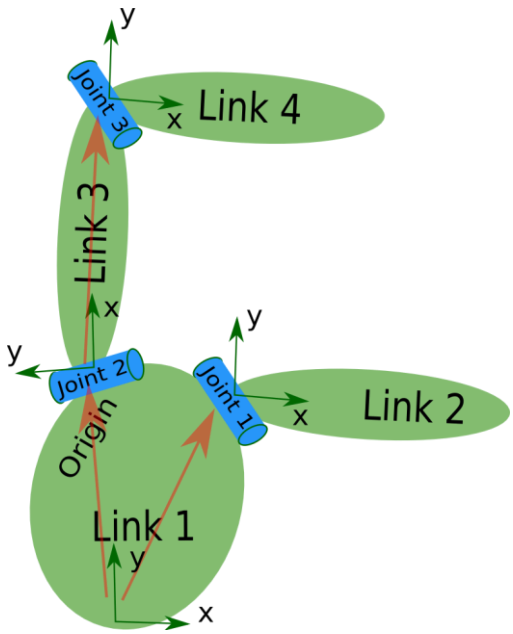

Figura 17: Esquema URDF.

La organización EuRoC nos facilita el archivo URDF de nuestro robot UR5, que utilizaremos para realizar las operaciones de cinemática inversa y directa usando la librería *ur_kin_py* , que detallamos a continuación, ya que para realizar operaciones de cinemática necesitamos conocer el modelo cinemático del robot, y este archivo URDF le servirá a nuestra librería todo lo necesario sobre eslabones y articulaciones de nuestro manipulador. Para una información más completa

4.6.5 UR_KIN_PY

UR_KIN_PY [8] se trata de un paquete existente en el repositorio de códigos de ROS donde encontramos las funciones de cinemática directa e inversa de las configuraciones UR5 y UR10, que necesitaremos para la generación de trayectorias para mover nuestro robot.

4.6.6 Mercurial

Ciertamente Mercurial no pertenece al sistema ROS, es un software propio y totalmente fuera, es un sistema de control de versiones multiplataforma, un software libre para la línea de comandos, implementado principalmente en Python pero que incluye una implementación binaria en C. Escrito originariamente para funcionar en GNC/Linux. Con una interfaz web integrada, sus metas de desarrollo son el rendimiento y escalabilidad, un desarrollo distribuido sin necesidad de servidor, gestión robusta de archivos tanto de texto como binarios y capacidades avanzadas de ramificación e integración manteniendo la sencillez conceptual. Las diferentes facilidades que nos ofrece Mercurial son:

- Monitorizar los archivos añadidos
- Repositorio compartido
- Guardar cambios localmente de forma temporal trabajando en diferentes clones
- Copiar y mover archivos
- Revisar historial
- Arreglar errores en versiones anteriores
- Mezcla o fusión de dos clones diferentes originales de un mismo código

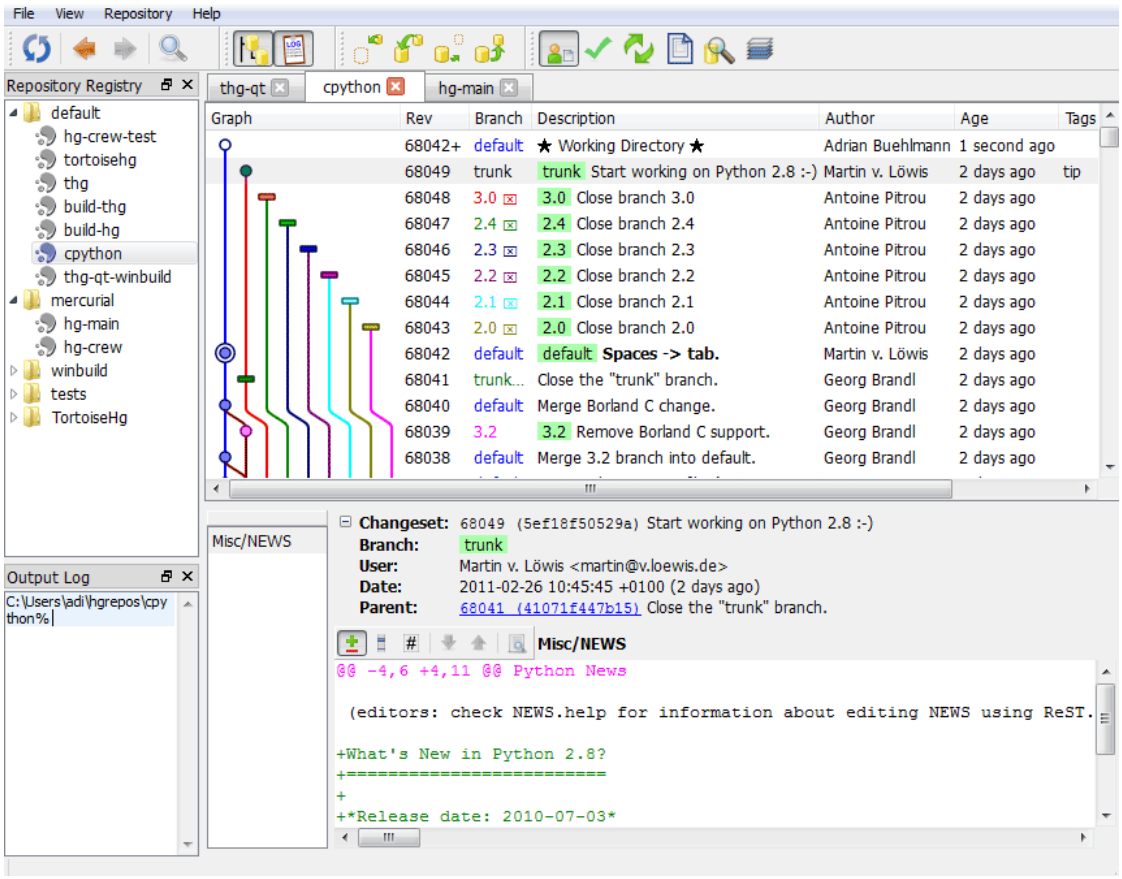

Figura 18: Mercurial.

5 ENTORNO DE SIMULACIÓN EUROC

Como ya se desarrolló anteriormente, mi proyecto se encontraba en la prima fase del Challenge 1 de EuRoC, dónde se planteaban 4 tareas en dos escenarios distintos, las tareas 1 y 2 en uno, y la 3 y 4 en otro. A continuación mostraremos una descripción del entorno en el que reside el trabajo, escenario uno, para la tarea 2.

5.1 Entorno de simulación

El entorno para las tareas 1 y 2 consiste en área de trabajo donde encontramos principalmente una mesa de trabajo, con un robot industrial y un objeto, un operario moviéndose de forma totalmente aleatoria (en nuestra tarea 2) por nuestro escenario, y dos cámaras 3D.

Sobre la mesa de trabajo encontramos un robot industrial de 6 grados de libertad (configuración UR5) equipado en su efector final con una herramienta de remachado. También tiene lugar un objeto, que llamaremos de aquí en adelante "pieza de trabajo" en el área de trabajo de nuestro robot. Esta pieza de trabajo tiene varios agujeros para realizar operaciones de remachado sobre ellos.

Como ya hemos mencionado, también tenemos dos cámaras 3D. La primera enfoca el escenario de forma global pudiendo ver la escena con su totalidad, y la segunda enfoca directamente la mesa de trabajo. Cabe mencionar que en nuestra tarea 2 no usaremos estas cámaras, ya que éstas están destinadas para su uso en la tarea 1, centrada en la percepción.

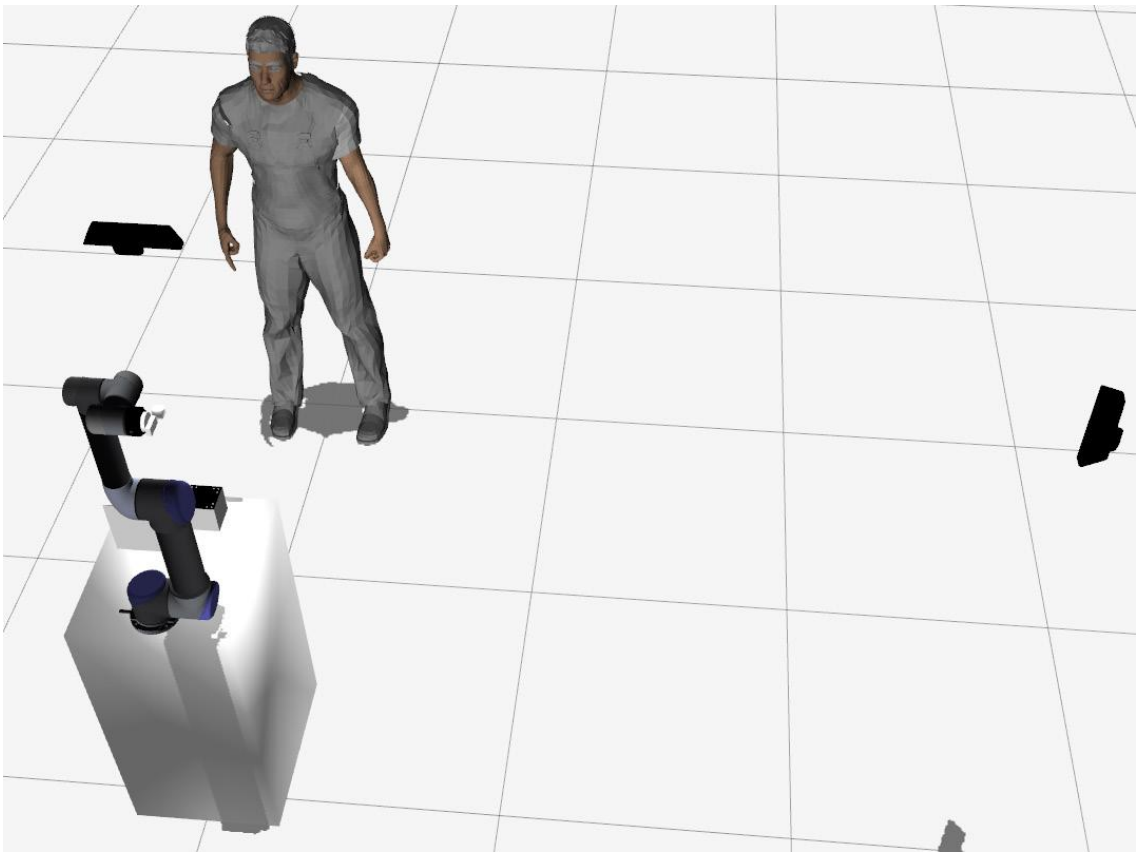

Figura 19: Entorno de simulación.

5.1.1 Sistema robótico

Como ya iniciamos antes, para nuestra tarea tenemos un robot manipulador de 6 grados de libertad, cuya configuración es la conocida y global UR5. La organización EuRoC nos da el acceso a su descripción cinemática y dinámica a través de su archivo URDF (Unified Robot Description Format).

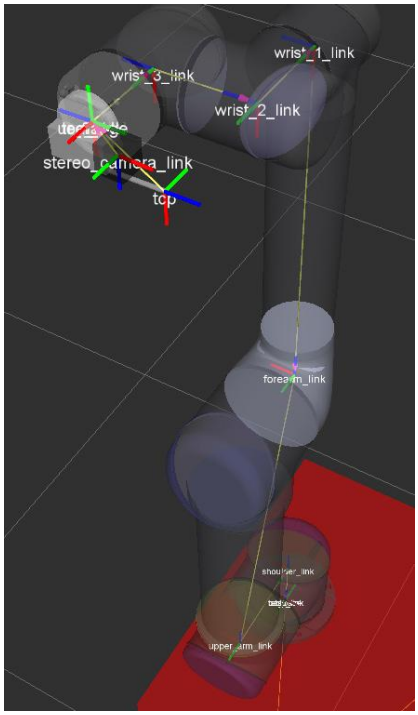

Figura 20: Robot UR5 con sus marcos de referencia.

En su efector final, el robot está equipado con una herramienta de remachado, descrita con la siguiente figura:

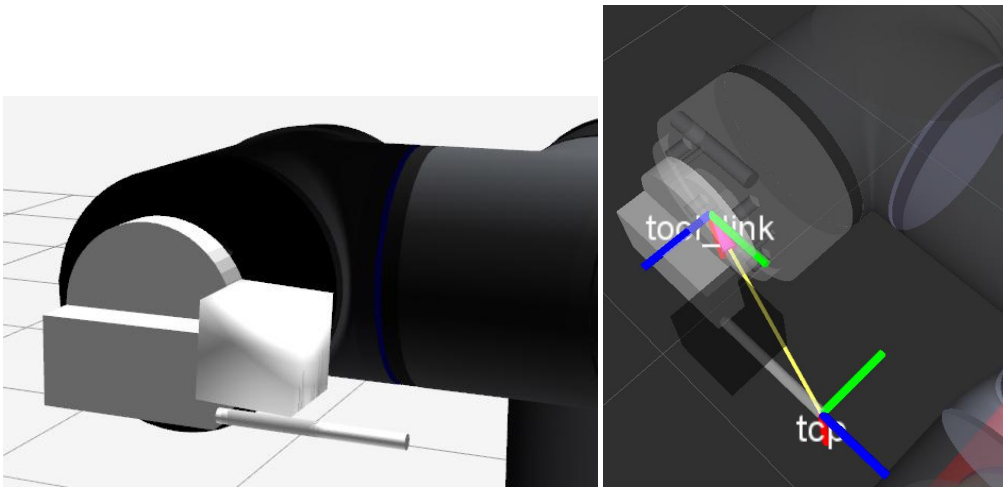

Figura 21: Vistas detalladas del efector final del robot.

Esta parte cilíndrica de 5mm de diámetro de la herramienta tiene que ser insertada en los agujeros de la pieza de trabajo, como detallaremos en la descripción de la tarea, los TF más relevantes se muestran en la imagen anterior. Debemos destacar que nuestra guía de la posición del robot será el último tf frame llamado *tcp*.

5.1.2 Humano simulado

En nuestro entorno, contamos con un modelo humano cuya interferencia será esencial en el desarrollo de las tareas, especialmente en la tarea 1, dedicada a la percepción de gestos del humano. El operario puede moverse libremente por el escenario y mover sus brazos a través de sus hombros, donde tiene una articulación.

Figura 22: Humano simulado.

5.1.3 Pieza de trabajo

La pieza de trabajo está localizada en la mesa enfrente del robot y contiene 33 agujeros. Se usará en las tareas de dos formas distintas:

- Tarea 1: Debe reconocerse el punto exacto de la pieza de trabajo que el operario señala.
- Tarea 2: Sobre sus agujeros debemos realizar las operaciones de remachado.

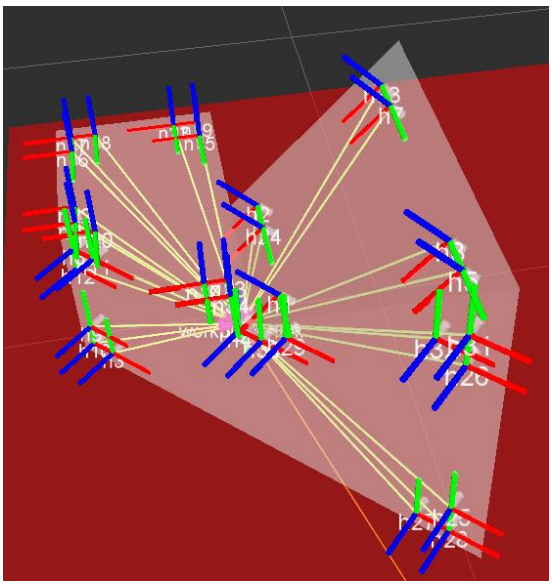

Figura 23: Pieza de trabajo.

Aquí podemos ver las dimensiones de la pieza:

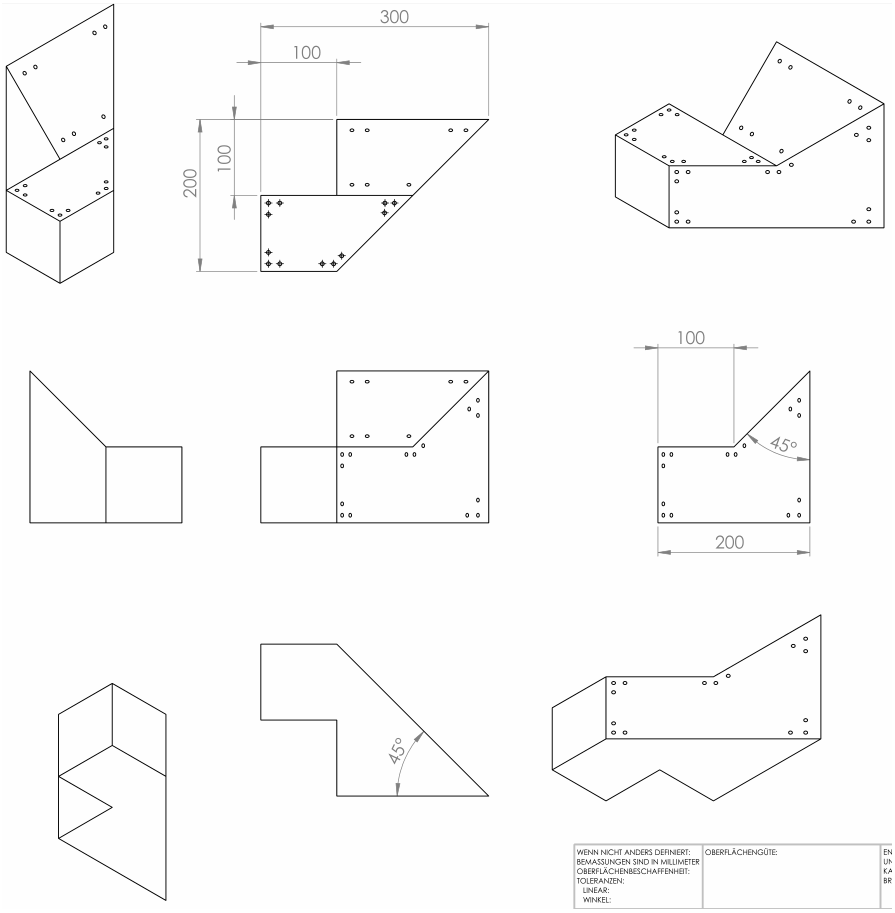

Figura 24: Dimensiones de la pieza de trabajo.

5.2 Objetivo de la tarea

El objetivo principal de esta tarea, y proyecto en cuestión, es realizar una serie de remachados en un tiempo especificado, con algunas especificaciones concretas durante la ejecución.

El proceso de ejecución será de 300 segundos, donde debemos hacer remachados de forma continua, con algunas instrucciones impuestas por la organización EuRoC, en función del topic *target_frame*.

El topic *target_frame* puede tomar dos valores, *rivet_home* y *hole_guess.*

Inicialmente, *target_frame == rivet_home*, y cada vez que *target_frame* tome este valor debemos dirigirnos al posición dada por el *tf rivet_home*, es decir, debemos hacer coincidir tanto en posición como orientación el tf del robot *tcp* con *rivet_home*. Una vez posicionado en rivet_home, debemos llamar al servicio *load_rivet*, que representa la recarga de la herramienta de remachado.

Una vez estamos en *rivet_home* y haber llamado al servicio *load_rivet* estamos listos para realizar la operación de remachado. Cuando movemos nuestro robot hacia esta posición, *target_frame* cambia automáticamente al valor *hole_guess*, indicándonos que debemos localizar que agujero remachar. Notar que hasta que la posición *tcp* no sea la misma que *rivet_home, target_frame* no cambiará, y no podremos continuar con la operación.

El tf frame *hole_guess* se actualizará de forma cíclica cada vez que *target_frame* cambie el valor de *hole_guess* a *rivet_home,* y se generará de forma aleatoria cerca de cualquiera de los diferentes agujeros que tiene la pieza de trabajo.

Una vez detectado el agujero a remachar, nos moveremos hacia él, e introduciremos la herramienta hasta un

máximo de 6 mm sin tocar la pieza de trabajo en ningún momento. Una vez la herramienta insertada, llamaremos al servicio *trigger_rivet*, que representa el remachado, y ya estaremos listos para el siguiente ciclo. Al igual que ocurre con *rivet_home*, *target_frame* no cambiará de valor de *hole_guess* hasta que el tf *frame hole_guess* no coincida con el *tf frame* del robot final *tcp*.

El topic target_frame cambiará de nuevo a *rivet_home*, donde deberemos ir de nuevo, y realizar el proceso descrito de forma cíclica en el tiempo de ejecución dado.

El topic *collision_workpiece_robot* indica colisiones entre la pieza de trabajo y el robot, y sirve a la organización para la evaluación de la solución.

En resumen, los principales objetivos de la tarea son:

- Mover el robot a través de las herramientas provistas de ROS.
- Identificar y localizar el agujero a remachar de forma cíclica.
- Realizar de forma exitosa los remachados evitando el contacto robot-pieza de trabajo durante el tiempo de ejecución dado.

5.3 Interacción con el sistema

Durante la ejecución de la tarea podemos hacer uso de la comunicación de distintos canales y fuentes de información de ROS que nos da EuRoC.

- Podemos obtener información de los siguientes topics:

TOPIC	**TIPO**	**DESCRIPCIÓN**
target_frame	std_msgs.msg.String	Nombre del tf frame donde debemos ir. Tomará los valores *rivet_home* o *hole_guess*
collision_workpiece_robot	std_msgs.msgs.Bool	Tomará valor True si hay contacto entre robot y pieza de trabajo
time_budget		Cuenta atrás de tiempo de ejecución de la tarea

Tabla 1: Topics disponibles en la ejecución.

- Podemos obtener información de los siguientes tf frames:

TF	**DESCRIPCIÓN**
rivet_home	Posición a la que debemos ir después de cada proceso de remachado.
hole_guess	Posición aproximada del agujero. Cambia en cada ciclo.
workpiece_link	Posición de la pieza de trabajo
h1, h2…	Información de la posición de cada agujero de la pieza de trabajo.

Tabla 2: TF frames disponibles en la ejecución.

- Podemos llamar a los siguiente servicios de ROS:

SERVICIO	DESCRPCIÓN
load_rivet	Llamaremos a este servicio cuando nos encontremos en *rivet_home*.
trigger_rivet	Llamaremos a este servicio cuando hayamos insertado la herramienta de remachado.

Tabla 3: Servicios ROS disponibles en la ejecución.

- Podemos llamar a las siguientes acciones de ROS:

ACCIÓN	DESCRIPCIÓN
/simulation_ur5/joint_trajectory_controller/follow_joint_trajectory	Usaremos esta acción para mover nuestro robot en el espacio articular.
/simulation_interface_ur5/cartesian_position	Usaremos esta acción para mover nuestro robot en el espacio cartesiano.

Tabla 4: Acciones ROS disponibles en la ejecución.

5.4 Evaluación

La organización EuRoC nos valorará en la solución de la tarea como ya hemos dicho, el número de remachados durante el tiempo de ejecución, la precisión de la estimación de agujeros a remachar y la ausencia de colisiones robot-pieza de trabajo. La evaluación se realizará sobre 30 puntos en la tarea, repartidos del siguiente modo:

SUBTAREA	INTERVALO DE PUNTUACIÓN	PUNTUACIÓN DADA
Media de error de localización de agujeros a remachar	Error superior a 5 mm o 5º	0
	Error entre 1.5 y 5 mm o 3º a 5º	2
	Error entre 0.3 y 1.5 mm o 1º a 3º	5
	Error inferior a 0.3 mm o 1º	10
Número de remachados realizados correctamente	0-4 remachados	0
	5-9 remachados	2
	10-14 remachados	5
	15 o más remachados	10
Número de colisiones robot-pieza de trabajo	>0 colisiones	0
	0 colisiones	10

Tabla 5: Evaluación de la tarea.

6 ALGORITMO DE PLANIFICACIÓN Y CONTROL DE PROCESO DE REMACHADO

En este capítulo expondremos la solución que se propuso y se desarrolló para la tarea, dividida en subproblemas de más fácil resolución, para ir escalando modularmente y resolver el problema por completo.

Para iniciar la tarea, es necesario arrancar dos nodos, un nodo servidor que nos proporciona toda la infraestructura y entorno necesario que dispone EuRoC, y un nodo cliente que es nuestra misión de realizar (programar) para que en su ejecución resuelta nuestra tarea, todo esto está tratado con mayor ampliación en el Anexo A "Arranque del sistema".

Una vez planteado el problema a resolver, decidí dividir la tarea en distintos subproblemas, en los cuáles su desarrollo me acercaría hasta la resolución final requerida, estos subproblemas son:

- Planificar cíclicamente la secuencia de remachado durante la ejecución: En este caso la decisión fue la realización de una máquina de estado secuencial.
- Mover el robot: ¿Cómo moveríamos el robot y qué necesitaríamos? Partimos de las opciones que nos facilita EuRoC, ROS actions para mover el robot tanto en el espacio articular como cartesiano.
- Detectar cíclicamente el agujero a remachar: Ya que en cada secuencia, debemos localizar y publicar cuál es el agujero de la pieza de trabajo dónde debemos realizar el remachado cíclicamente.

La resolución de la tarea debe realizarse mediante la ejecución del nodo cliente, por lo que se trata de programación, en este caso, a mi elección, utilicé el lenguaje de programación Python, por su claridad y sencillez frente a otros lenguajes que nos permite usar ROS.

A continuación detallaremos la resolución de los subproblemas en los que dividimos la tarea.

6.1 Máquina de estado

Por simplicidad y debido al carácter secuencial de la tarea, decidí implementar una máquina de estado en el nodo cliente, para la ejecución principal. Nuestra máquina de estado está compuesta por 7 estados diferentes y sus respectivas transiciones, generalmente, nuestras transiciones serán esperas que se activan mediante variables booleanas que permiten saltar al estado siguiente.

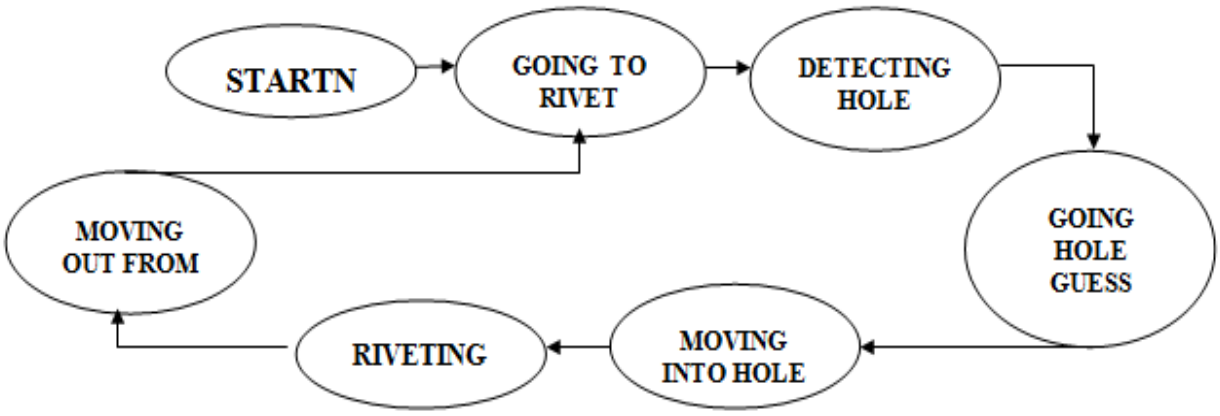

Figura 25: Diagrama de Estados.

A continuación haremos una breve descripción de cada estado y sus transiciones:

- STARTING: Es el estado inicial al arrancar la infraestructura, con el robot en una posición inicial dada por EuRoC.
- STARTING->GOING TO RIVET HOME: Esta transición se activará cuando *target_frame=rivet_home*, es decir, cuando este topic nos indique que debemos ir a la posición "inicial" de cada ciclo rivet home.
- GOING TO RIVET HOME: Se trata del estado de movimiento hasta *rivet_home*. Una vez lleguemos, debemos llamar al servicio *load_rivet*, tal como se nos indica que debemos hacer.

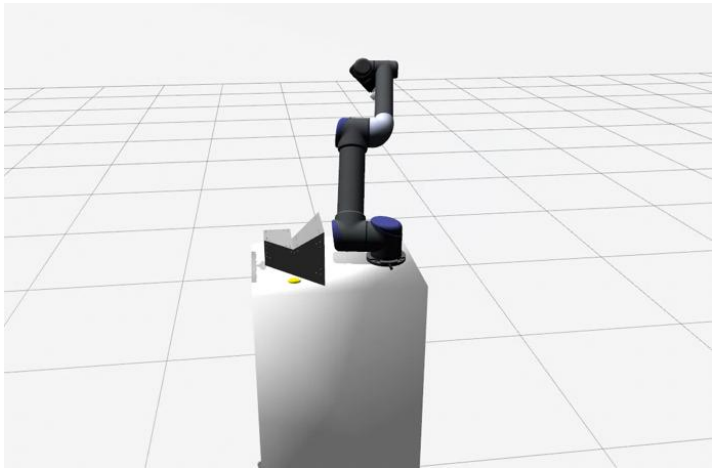

Figura 26: Posición "rivet_home" inicial del robot.

- GOING TO RIVET HOME -> DETECTING HOLE GUESS: Esta transición se activará cuando al llegar nuestro robot a rivet_home, automáticamente *target_frame* cambie su valor a *hole_guess.*
- DETECTING HOLE GUESS: Aquí hacemos llamada al a función *"detecting_hole_guess"*, detectando el agujero a remachar del ciclo y obteniendo las posiciones de la trayectoria que vamos a seguir para el remachado.
- DETECTING HOLE GUESS -> GOING HOLE GUESS: Esta transición se realiza de forma directa a través de una variable booleana.
- GOING TO HOLE GUESS: Donde realizaremos el movimiento del robot para encarar la pieza de la cara del agujero que queremos remachar.

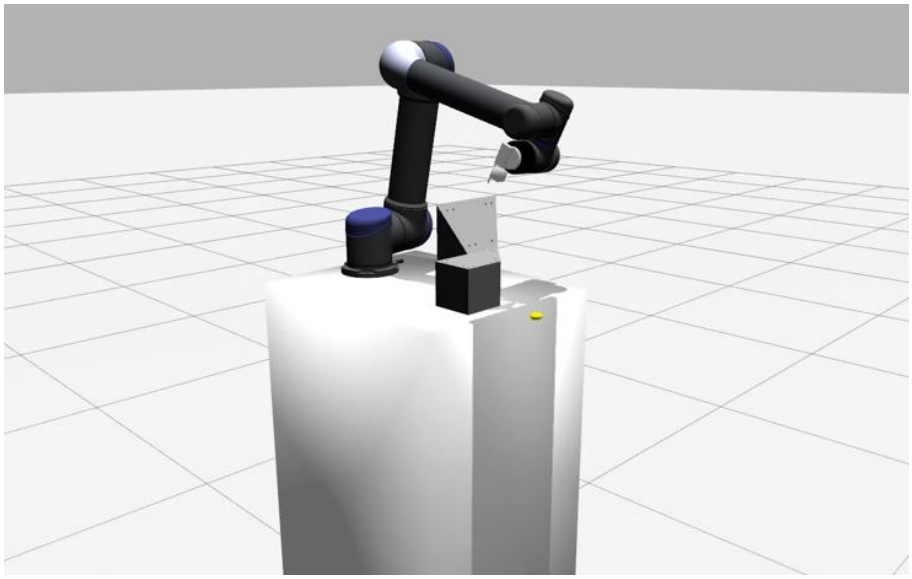

Figura 27: Posición intermedia de remachado.

- GOING TO HOLE GUESS -> MOVING INTO HOLE: De 36utom transición automatic mediante variable booleana
- MOVING INTO HOLE: Donde nuestro robot se introducirá en el agujero hasta 6 mm con la herramienta de remachado cilíndrica.

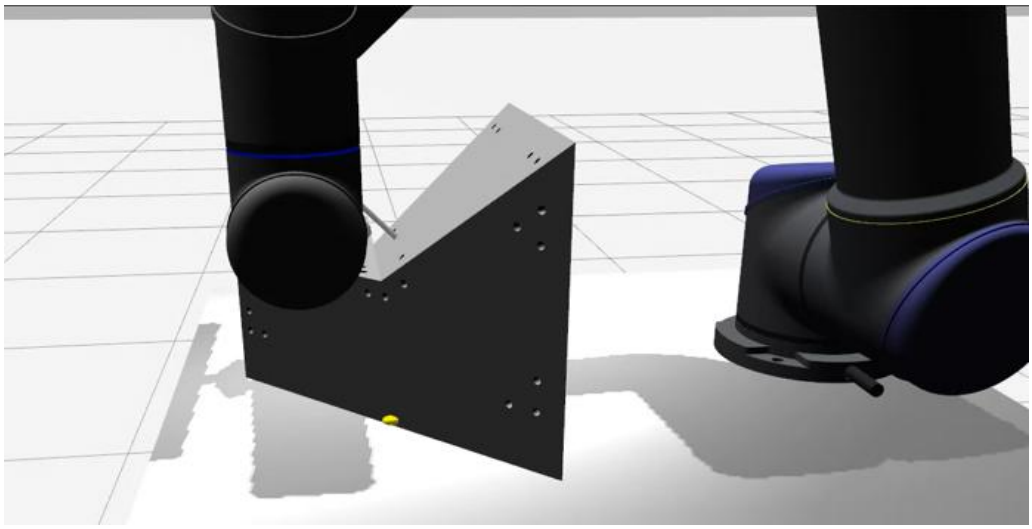

Figura 30: Remachado de la pieza.

- MOVING INTO HOLE -> RIVETING: Transición 36utomatic por variable booleana.
- RIVETING: Una vez introducida la herramienta en el agujero, llamaremos al servicio ROS *trigger_rivet*, que simula el proceso de remachado, y podremos contabilizar un remachado con éxito.
- RIVETING -> OUT FROM HOLE: Transición que se active tras la llamada al servicio *trigger_rivet.*
- OUT FROM HOLE: Movimiento del robot de salida del agujero hasta la posición de encare inicial.
- OUT FROM HOLE -> GOING TO RIVET HOME: Si todo se ha realizado correctamente, nuestro topic *target_frame* tendrá un valor de *rivet_home* tras comprobar esto, se activará la transición, comenzando el proceso de nuevo.

Esta máquina de estado está implementada en código de programación que se adjunta en el Anexo B :"State Machine Code". Este ejecutable está implementado en el nodo cliente de la tarea, donde debe ir la solución a nuestro problema.

6.2 Mover el robot

Como expuse anteriormente, partimos de la base de opciones de acciones en ROS que nos proporciona la organización del concurso:

- *simulation_ur5/joint_trajectory_controller/follow_joint_trajectory:* Nos permite mover el robot en el espacio articular.
- *simulation_interface_ur5/cartesian_position*:ipa325_msgs.msg.RobotMovementAction : Nos permite mover el robot en el espacio cartesiano.

Tras una serie de pruebas utilizando ambas opciones, finalmente me decanté por mover nuestro robot en el espacio articular, ya que la posibilidad del espacio cartesiano nos ofrecía a veces problemas de cinemática inversa, produciendo movimientos bruscos del robot, golpeando la pieza de trabajo, cosa que debemos evitar.

Tratándose de una acción de ROS, debemos saber que se trata de una comunicación bidireccional, es decir, debemos hacer una petición de realizar la acción, esta petición se realizar mediante una llamada de código. Una vez formulada la petición, si ésta se ha realizado de forma correcta, se ejecutará la acción, y nos enviará un mensaje de vuelta con el resultado: éxito, error o indefinido. Puede entenderse tal que así:

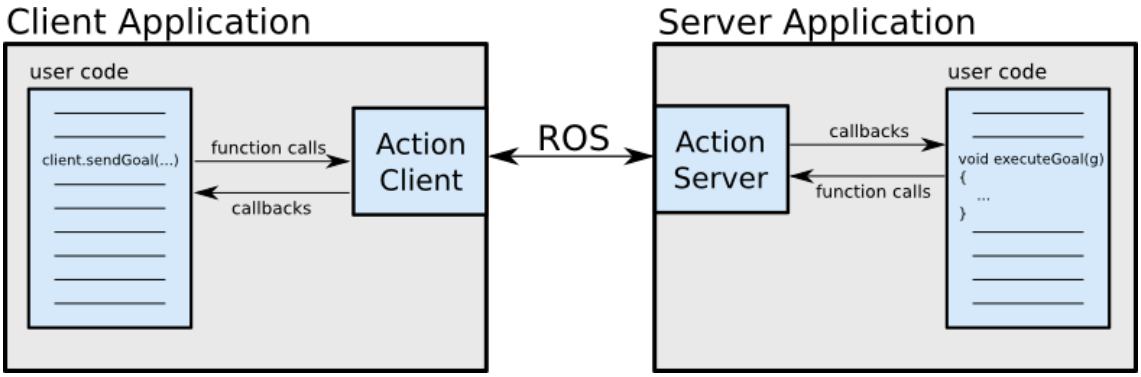

Figura 31: Comunicación entre acciones ROS.

Para realizar nuestra petición a la acción, debemos completar una estructura de mensaje ROS, del tipo *control_msgs.msg.FollowJointTrajectoryGoal*, donde debemos completar los siguientes campos de la estructura *control_msgs.msg.FollowJointTrajectoryGoal.trajectory*:

- *control_msgs.msg.FollowJointTrajectoryGoal.trajectory.joint_names*: Se trata de un vector de caracteres compuesto por las distintas articulaciones de nuestro robot, en nuestro caso , tras definir nuestra variable de mensaje:

  goal_trajectory = control_msgs.msg.FollowJointTrajectoryGoal();
  goal_trajectory.trajectory.joint_names = [‘shoulder_pan_joint’, ‘shoulder_lift_joint’, ‘elbow_joint ‘wrist_1_joint’, ‘wrist_2_joint’, ‘wrist_3_joint’];

- *control_msgs.msg.FollowJointTrajectoryGoal.trajectory.points*: Donde definíamos la trayectoria de puntos de nuestro movimiento en el espacio articular, dentro de *points* debíamos definir dos subcampos:

  - *control_msgs.msg.FollowJointTrajectoryGoal.trajectory.points.positions:* Valor articular de nuestra trayectoria, en nuestro caso, era válido enviarle simplemente posición inicial y final
  - *control_msgs.msg.FollowJointTrajectoryGoal.trajectory.points.time_from_start.secs:* Definiendo así la velocidad del movimiento, es decir, el instante en el que queríamos que nuestro robot estuviese en cada posición.

Una vez completos estos campos realizábamos la petición de acción y movíamos el robot de forma satisfactoria.

Pero antes de la petición había que realizar un trabajo previo, ya que inicialmente todas nuestras posiciones destino las obteníamos mediante la herramienta TF en valores cartesianos, por lo que hay que aplicar funciones de cinemática inversa para poder continuar con el subproblema.

Esto constituyó uno de los principales problemas para la resolución del proyecto, ya que inicialmente se decidió usar la librería de ROS KDL, la más generalizada y usada para la resolución de problemas cinemáticos tanto directos como inversos, pero en nuestro caso nunca dio resultados satisfactorios, ya que realizaba los movimientos con poca precisión, por lo que no nos era válido debido a que estos errores de posición daban lugar a contacto robot-pieza de trabajo, algo que debíamos evitar a toda costa.

Finalmente, se decidió utilizar la librería ur_kin_py (explicada en capítulos anteriores) para este problema, realizando las funciones de cinemática inversa de manera satisfactoria.

Todo esto se implementó en la función "*move_to"* que usamos en la máquina de estado, donde la función recibe la posición destino, y se encargará de realizar todo lo necesario descrito anteriormente.

En resumen, para el movimiento del robot era necesario:

1. Conocer, mediante la herramienta ROS TF, la posición inicial y final en coordenadas cartesianas de nuestro movimiento deseado.
2. Transformar estos valores cartesianos en articulares aplicando cinemática inversa.
3. Rellenar la estructura de mensaje de la petición de acción ROS.
4. Petición de acción.
5. Esperar y obtener resultados.

Siguiendo este esquema, se consiguió de manera satisfactoria mover nuestro robot, obteniendo siempre buenos resultados y resolviendo nuestros problemas iniciales con la cinemática inversa con la librería que finalmente usamos.

6.3 Detectar agujero de remachado

Durante la ejecución de la tarea, nuestro agujero a remachar de forma cíclica no se publica de forma explícita, sino a través de un TF frame llamado *hole_guess* que se publica automáticamente en cada ciclo, y que es una aproximación del verdadero agujero de la pieza a remachar, y que debemos identificar y localizar nosotros desde la posición inicial, que es uno de los objetivos de nuestra tarea.

Nuestra pieza de trabajo está compuesta por un total de 33 agujeros, distribuidos en tres caras o superficies distintas de la pieza. Para localizar nuestro agujero a remachar, decidí usar la herramienta de ROS TF, que nos permite obtener posiciones relativas entre distintos TF frame.

Nuestra función "*detecting_hole_guess"*, de forma general, realiza un bucle recorriendo los 33 agujeros de la pieza, comparando cada uno de sus TF frames con el TF frame *hole_guess*, buscando así, el agujero que minimice el valor absoluto de la comparación en posición y orientación. Ésta nos devolvería una cadena de caracteres indicando de qué agujero se trata, para realizar la operación de remachado sobre él en ese ciclo.

```
For I in xrange(33):
        h_i = "h"+str(i)
        hi_pos = self.get_tf_transform("base_link", h_i)
        pos_dif = numpy.array(hi_pos)-numpy.array(hole_guess_pos)
        result_i = numpy.linalg.norm(pos_dif)
        if result_i<result:
           h_result="h"+str(i)
           hole_numb = i

     if hole_numb == -1:
        rospy.loginfo("Hole_guess not detected")
        q_home = -1

     else:
        rospy.loginfo("Hole_guess detected: ")
        rospy.loginfo(h_result)
```

Además, con vistas a la eficiencia de la ejecución de nuestro algoritmo, se que según la cara en la cual se encontrase nuestro agujero (conocemos cuál agujero pertenece a cuál cara), realizaríamos una trayectoria diferente con nuestro robot, es decir, encararíamos a la pieza de forma diferente, con el fin de realizarlo con mayor velocidad y eliminar posibles errores de cinemática inversa en nuestra trayectoria. Por lo que en nuestra función, además de localizar el agujero a remachar, nos devuelve en valores articulares las distintas posiciones de nuestra trayectoria para realizar la operación en función de la cara de la pieza donde se encuentre el agujero. A continuación mostramos imágenes sobre los "encaramientos" de nuestro robot hacia la pieza:

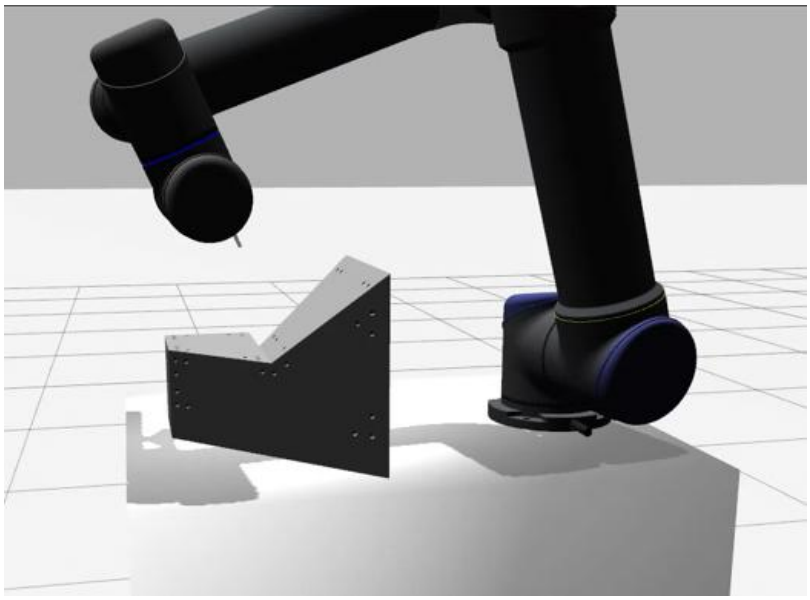

Figura 32: Posición intermedia de remachado para cara superior.

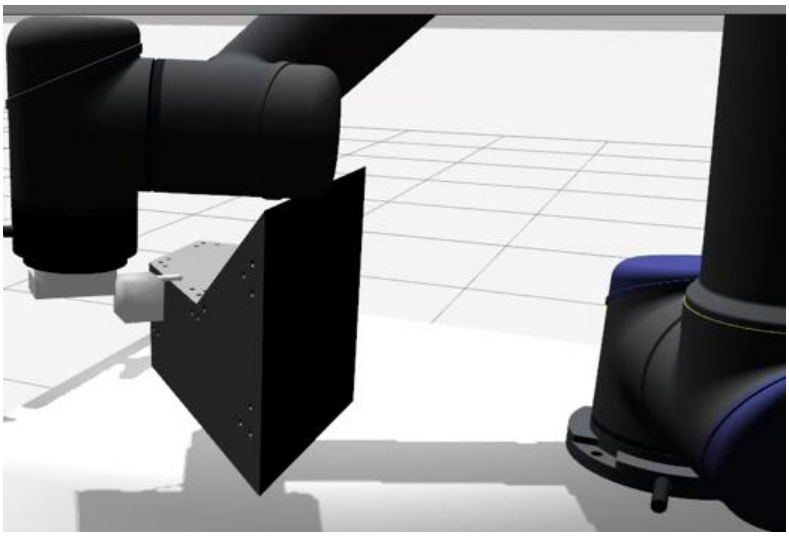

Figura 33: Posición intermedia de remachado para una cara lateral.

6.4 Ejemplo de ejecución

A continuación mostraremos una serie de imágenes que nos visualiza un ejemplo de ciclo de remachado de nuestro robot:

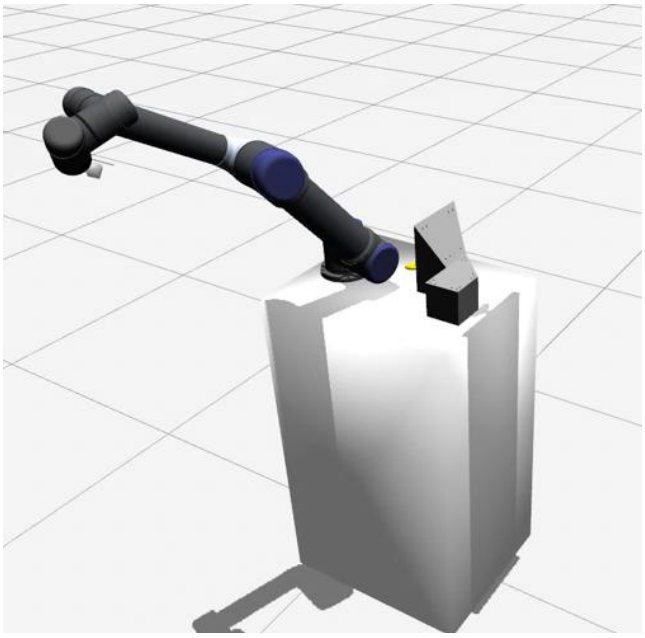

Figura 34: Posición inicial “rivet_home”.

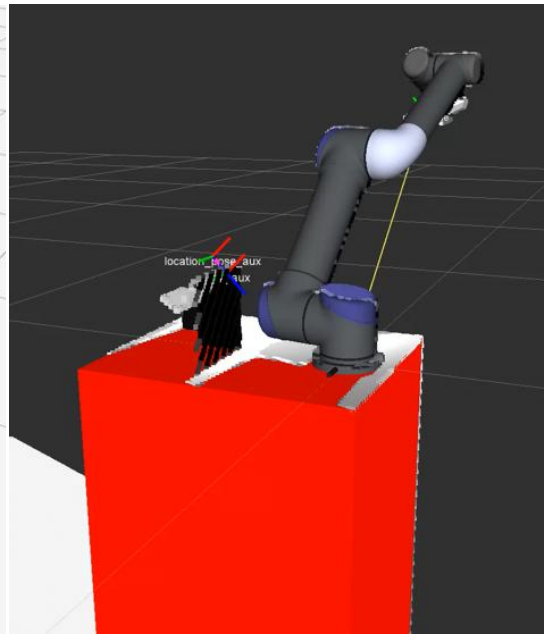

Figura 35: Robot detectando “hole_guess”.

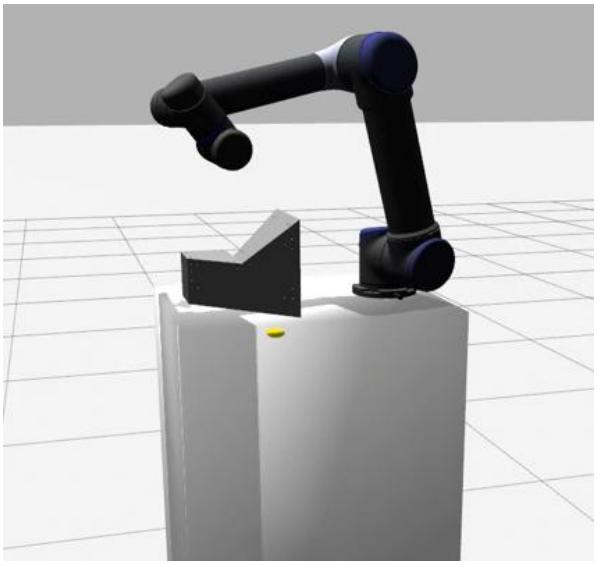

Figura 36: Robot encarando pieza

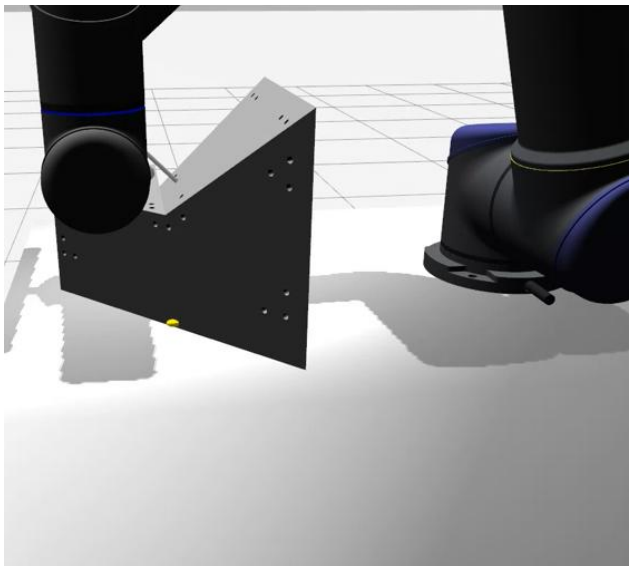

Figura 37: Inserción de la herramienta de remachado

A continuación, volveríamos a la posición inicial y volveríamos a buscar de nuevo el siguiente “hole_guess”.

7 RESULTADOS Y CONCLUSIONES

Con la solución propuesta al problema expuesta en el capítulo anterior, tras la evaluación de la organización EuRoC, los resultados de la tarea fueron los siguientes tras su ejecución:

SUBTAREA	MEDIDA	RESULTADO	PUNTUACIÓN OBTENIDA
Localización de agujeros a remachar	Media de error	0	**10**
Remachar agujeros	Número de agujeros remachados correctamente	9	**2**
Colisiones robot-pieza de trabajo	Número de colisiones robot-pieza de trabajo	0	**10**

Tabla 6: Resultados de evaluación.

Dentro de la tarea del proyecto, obtuvimos un total de 22/30 puntos, y englobando todas las tareas de la primera fase del concurso, obtuvimos un total de 43 puntos, 21 de la tarea 1, 22 de la presente, y 0 puntos tanto para la tarea 3 y 4, ya que finalmente no presentamos ninguna solución para esta parte de la primera fase del Challenge 1. Puede ver la tarea 2 en el formato digital aquí

Esta puntuación nos permitió para clasificarnos para la segunda fase del Challenge 1, obteniendo un octavo puesto de los 32 participantes (recordamos que solo se clasificaban los 15 primeros clasificados), que ya quedó en manos del Centro Tecnológico CATEC y en el que no participé.

A la vista de los resultados en la evaluación, nuestro punto flojo fue la rapidez con la que realizábamos cada ciclo de remachado, pero formaba parte de la estrategia de optimización de obtención de puntos, ya que como desarrollamos en el capítulo anterior, en el desarrollo, no fue posible darle mayor velocidad a la trayectoria del robot de manera estable sin obtener colisiones entre el robot y la pieza de trabajo, por lo que se decidió asegurar los 10 puntos referente al número de colisiones frente al número de remachado de agujeros, ya que aunque aumentásemos la velocidad del robot en sus movimientos y tuviésemos algunas colisiones (que esto nos daría directamente 0 puntos en la subtarea de colisiones) habría que realizar un total de 15 o más remachados de forma correcta para obtener los 10 puntos de esa subtarea, cosa que no se consiguió, por lo que se optó por la estrategia presente.

Por otra parte, mediante la herramienta TF de ROS conseguimos obtener de forma exitosa todas las localizaciones de los agujeros que había que remachar, teniendo un error nulo en cada uno de ellos.

Si hubiésemos conseguido una herramienta más eficiente para cálculo de cinemática inversa, ya que con la que disponíamos debíamos realizar trayectorias predefinidas que nos hacían movernos cíclicamente de forma más lenta, hubiésemos podido planificar trayectorias más directas entre posición inicial y posición de remachado y ganar tiempo, pudiendo realizar un mayor número de remachados en el tiempo de ejecución dado.

ANEXO A: ARRANQUE DEL SISTEMA

Para iniciar nuestra infraestructura proporcionada por EuRoC y la resolución al problema propuesto, debemos instalar previamente los paquetes desde el servidor de la organización, una vez instalados debemos distinguir dos extremos de comunicación:

- Nodo servidor: En su ejecución inicializa todo lo necesario para comenzar la tarea, es decir, el entorno de simulación en gazebo (los modelos visuales de todos los elementos del escenario), topics, servicios, TF frames… etc. Una vez inicializado todo, se mantiene a la espera de la ejecución del nodo cliente, que resolverá la tarea.
- Nodo cliente: Contiene el ejecutable para la resolución de la tarea, es decir, la llamada a la función de la máquina de estado, una vez lo arrancamos, resuelve la tarea automáticamente.

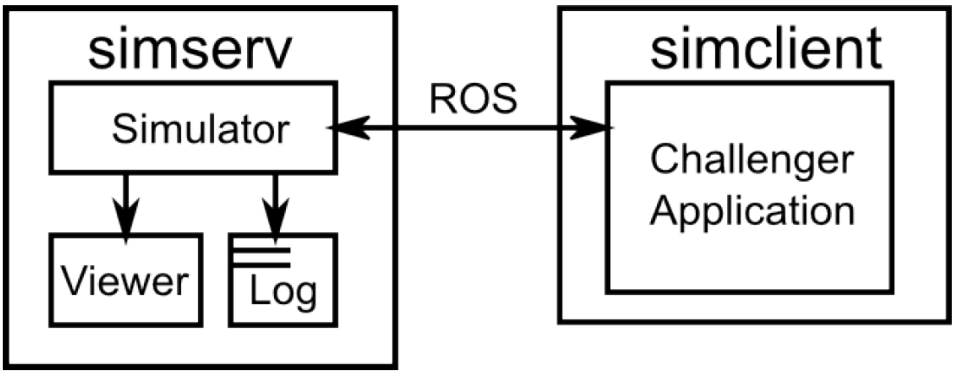

Figura 38: Esquema de comunicación servidor-cliente.

Para arrancar el nodo servidor debemos hacer un *roslaunch* en la consola Linux o sistema operativo usado*:*

roslaunch ipa325_euroc_sim track1.launch euroc_test_com_server:=False

Este *launch* contiene el nodo servidor llamado *ipa325_com_server.py*, una vez lanzado el *launch*, se nos inicializará toda la estructura y permanecerá a la espera cliente, que también se llama por medio de *roslaunch:*

roslaunch ipa325_com_client ipa325_com_client.launch

De Nuevo permanecerá en stadby la ejecución esperando que le indiquemos que tarea vamos a resolver, esto se lo indicaremos a través de un servicio ROS *rosservice*, que identificará qué tarea queremos resolver e irá a la parte del código del nodo cliente relativa a la tarea seleccionada:

rosservice call solvetaskid:'2'

Una vez iniciado el servicio comenzará automáticamente la resolución de la tarea, que podremos seguir sus resultados a través de Gazebo o RVIZ de forma visual, o a través de los mensajes que van imprimiendo los nodos por pantalla en la consola

ANEXO B: STATE MACHINE CODE

```
from enum import Enum
import rospy
import control_msgs
import std_srvs
import std_msgs
import std_msgs.msg
from catec_t12_trajectories import trajectory_generation, robot_interface
from catec_t12_trajectories.robot_interface import RobotInterface
from catec_t12_trajectories import trajectory_generation;
from catec_t12_trajectories.trajectory_generation import TrajectoryGenerator
from hrl_geom.pose_converter import PoseConv
import tf
import numpy
import math
import pylab
import sys
import time
from catec_t12_trajectories.position_manager import PositionManager

class State(Enum):
   STARTING = 1
   GOING_RIVET_HOME = 2
   DETECTING_HOLE_POSE = 3
   GOING_HOLE_GUESS = 4
   MOVING_INTO_HOLE = 5
   RIVETING = 6
   MOVING_OUT_FROM_HOLE = 7

class DrillingWorkflow:
   def __init__(self):
      self.current_state = State.STARTING
      self.robot = RobotInterface(connect_server=True)
      self.trajectory_generator=TrajectoryGenerator()
      self.count_rivet_success=0

      #-------------------------------------------------------------
      self.location_pose_aux=None;
      self.rivet_aux=None;
      self.globalAt=1;
      self.positions_manager=PositionManager(self.trajectory_generator);
      self.pm=self.positions_manager;
      self.rivet_home_pos = None
```

```
        #Booleans variables to change state
        self.detecting_hole_guess_bool = None
        self.going_hole_guess_bool = None
        self.moving_into_hole_bool = None
        self.riveting_bool = None
        self.moving_out_from_hol_bool = None
        #-------------------------------------------------------------
        self.rivet_home=None;
        self.hole_guess=None;
        rospy.Subscriber("target_frame",std_msgs.msg.String,self.on_target_frame_message_received)

    #---- PERIODIC UPDATE FUNCTIONS INSIDE STATES ------------------

def update_starting_state(self):
        m = rospy.wait_for_message('target_frame', std_msgs.msg.String);
        target_frame = m.data;
            if target_frame == '/rivet_home':
                    self.change_state(State.GOING_RIVET_HOME)

    def update_going_rivet_home(self):
        m = rospy.wait_for_message('target_frame', std_msgs.msg.String);
        target_frame = m.data;

        if target_frame == '/hole_guess':
                self.change_state(State.DETECTING_HOLE_POSE)

    def update_detecting_hole_pose(self):
        if self.detecting_hole_guess_bool:
                    self.detecting_hole_guess_bool=False
                    self.change_state(State.GOING_HOLE_GUESS)

    def update_going_hole_guess(self):
        if self.going_hole_guess_bool:
                    self.going_hole_guess_bool=False
                   self.change_state(State.MOVING_INTO_HOLE)

    def update_moving_into_hole(self):
        if self.moving_into_hole_bool:
                    self.moving_into_hole_bool=False
                    self.change_state(State.RIVETING)

    def update_riveting(self):
        if self.riveting_bool:
                    self.riveting_bool=False
                   self.change_state(State.MOVING_OUT_FROM_HOLE)

    def update_moving_out_from_hole(self):
        if self.moving_into_hole_bool:
                    self.moving_into_hole_bool=False
                    self.change_state(State.GOING_RIVET_HOME)
```

```
#------- TRANSITIONS ----------------------------------------

def change_state(self, new_state):
        if self.current_state == State.STARTING and new_state==State.GOING_RIVET_HOME:
                rospy.loginfo("state starting");
                rospy.loginfo("state going_rivet_home")

                self.rivet_home_pos = self.pm.get_tf_transform("base_link","rivet_home");
                self.move_to(self.rivet_home_pos, "rivet_home")
                 self.robot.load_rivet();

        elif self.current_state == State.GOING_RIVET_HOME
         and new_state==State.DETECTING_HOLE_POSE:

                 rospy.loginfo("state detecting hole guess")
                   self.location_pose_aux, self.rivet_aux, q_int=self.pm.detecting_hole_guess()


                   if q_int == -1:
                            rospy.loginfo("State Machine blocked, trying to detected hole guess again")
                           self.change_state(State.DETECTING_HOLE_POSE)


                   else:
                            int_configuration_trajectory=[ self.robot.get_current_configuration(), q_int];

                           goal_joint_trajectory=self.trajectory_generator.convert_goal_trajectory_from_joint
                           _trajectory(int_configuration_trajectory,[0,1.0], At=self.globalAt)
                           self.robot.send_goal_trajectory(goal_joint_trajectory,"intermediate position")
                        rospy.sleep(2)

                            self.detecting_hole_guess_bool=True

        elif self.current_state == State.DETECTING_HOLE_POSE
        and new_state==State.GOING_HOLE_GUESS:

                   rospy.loginfo("state going hole guess")
                   self.move_to(self.location_pose_aux, "location_pose_aux")
                   self.going_hole_guess_bool=True

        elif self.current_state == State.GOING_HOLE_GUESS and new_state==State.MOVING_INTO_HOLE:

                 rospy.loginfo("state moving into hole")
                   self.move_to(self.rivet_aux, "rivet_aux")
                   self.moving_into_hole_bool=True

        elif self.current_state == State.MOVING_INTO_HOLE and new_state==State.RIVETING:

                   rospy.loginfo("riveting")
                   self.robot.trigger_rivet();
                   self.count_rivet_success = self.count_rivet_success + 1
                   rospy.loginfo("Riveting reached:")
                print(self.count_rivet_success)
```

```
                self.riveting_bool=True

            elif self.current_state == State.RIVETING and new_state==State.MOVING_OUT_FROM_HOLE:

                rospy.loginfo("moving out from hole")
                  self.move_to(self.location_pose_aux, "location_pose_aux")
                  self.moving_into_hole_bool=True

            elif self.current_state == State.MOVING_OUT_FROM_HOLE
            and new_state==State.GOING_RIVET_HOME:

                rospy.loginfo("going rivet_home again")
                self.rivet_home_pos = self.pm.get_tf_transform("base_link","rivet_home");
                self.move_to(self.rivet_home_pos, "rivet_home")
                self.robot.trigger_rivet();
                self.going_rivet_home_bool=True

            else:
                pass

            self.current_state = new_state

    #---- PERIODIC UPDATE FUNCTIONS INSIDE STATES ----------------

      def update(self):
        self.robot.update()

        if self.current_state == State.STARTING:
                self.update_starting_state()

        if self.current_state== State.GOING_RIVET_HOME:
                self.update_going_rivet_home()

        if self.current_state== State.DETECTING_HOLE_POSE:
                self.update_detecting_hole_pose()

        if self.current_state== State.GOING_HOLE_GUESS:
                self.update_going_hole_guess()

        if self.current_state== State.MOVING_INTO_HOLE:
                self.update_moving_into_hole()

        if self.current_state== State.RIVETING:
                self.update_riveting()

        if self.current_state== State.MOVING_OUT_FROM_HOLE:
                self.update_moving_out_from_hole
```

BIBLIOGRAFÍA

[1] EuRoC Organización: http://www.euroc-proyect.eu

[2] Spong, M., Hutchinson, S. (2005). *Robot Modeling and Control*.

[3] ROS: http://ros.org

[4] Gazebo: http://gazebosim.org

[5] RVIZ: http://wiki.ros.org/rviz

[6] TF: http://wiki.ros.org/tf

[7] URDF: http://wiki.ros.org/urdf

[8] UR Kin Py Tool: http://wiki.ros.org/ur_kin_py