

Universitat d'Alacant Universidad de Alicante

Grado en Ingeniería Robótica: Robots Móviles

Proyecto Final: LOGPAT (Logistic Patrol)

Andrés Antón Ruvira
Miguel Ángel Pascual Morante
Miguel Ángel Piqueres Fernández
Miguel Ferrer Castellá

Índice

1. Introducción.....	3
1.1. Idea del proyecto.....	3
2. Implementación de subtareass.....	5
2.1. Funcionamiento general de la máquina de estados.....	5
2.2. Lectura de etiquetado.....	6
2.3. Vuelta a base.....	7
2.4. Patrulla.....	7
3. Base de datos.....	9
3.1. Supabase.....	9
3.2 Gestión de Base de Datos.....	9
3.3 Interacción Manual.....	12
4. Vídeo de los resultados obtenidos.....	12
5. Bibliografía.....	12

1. Introducción

1.1. Idea del proyecto

El siguiente proyecto tiene como objetivo principal desarrollar e implementar un conjunto de algoritmos avanzados que permitan al robot móvil *SUMMIT-XL STEEL*, fabricado por la empresa española *Robotnik Automation*, con sede en Valencia, realizar un patrullaje autónomo, eficiente y seguro dentro de un entorno logístico, concretamente un almacén. Este proyecto se enmarca en la búsqueda de soluciones innovadoras para optimizar los procesos de gestión y control de inventarios mediante el uso de robótica móvil de última generación.



Figura 1. *SUMMIT-XL STEEL*, Robotnik.

El robot *SUMMIT-XL STEEL* desempeñará una serie de tareas clave durante su operación, diseñadas para mejorar la gestión y supervisión de los paquetes almacenados. Estas funciones incluyen:

Antecedentes

Dentro de este campo, destaca el papel del ingeniero y empresario Raffaello D'Andrea. Fue cofundador de Kiva Systems, una empresa que revolucionó la logística con robots móviles autónomos capaces de transportar estanterías de productos directamente a los trabajadores. Esta tecnología optimizó la gestión de pedidos, reduciendo tiempos y costos operativos. Tras la adquisición de Kiva Systems por Amazon en 2012, su trabajo sentó las bases de Amazon Robotics, que hoy gestiona millones de paquetes con sistemas inteligentes de coordinación de flotas.

Además de su papel en la robótica de almacenes, D'Andrea ha contribuido significativamente a la robótica aérea y la inteligencia artificial. Es fundador de Verity, una empresa que desarrolla drones autónomos para la gestión de inventarios, y ha trabajado en proyectos innovadores en el ETH Zürich, como drones acrobáticos y robots dinámicos. Su investigación en control y optimización ha influido en múltiples áreas, desde la robótica industrial hasta espectáculos de drones sincronizados, consolidándose como una figura clave en la automatización y la robótica moderna. A través del siguiente [enlace](#) pueden acceder a un video donde se detalla su trayectoria profesional.

Escaneo y Registro de Nuevos Paquetes

En una primera fase, en caso de que hayan llegados nuevos envíos a la planta logísticas, el robot los escaneará obteniendo los datos del etiquetado. Este proceso permite recopilar información esencial presente en la etiqueta, que será transferida automáticamente al sistema central de la empresa, donde se almacenan y gestionan de manera eficiente para garantizar un control preciso del inventario.

El formato de la etiqueta es un factor importante, se ha decidido que la empresa exige a sus proveedores un etiquetado específico, concretamente el modelo de etiqueta “*GS1 Non-Std HVM*” ampliamente utilizado en la industria logística. A continuación, se muestra un ejemplo de este tipo de etiqueta.

From: Logística Universal S.A.	To: Distribuciones del Norte S.L.
Calle Industria 45 Barcelona, España	Av. Central 89 Madrid, España
SSCC: 473598741002145201	COUNT: 15
CONTENT 8476543210789	NET WEIGHT (g): 3000
BATCH/LOT: B234567	USE BY (DD.MM.YYYY): 23.9.2029

Figura 2. Modelo *GS1 Non-Std HVM*.

Podemos ver que contiene información respecto a la dirección del remitente y destinatario, código SSCC, código de contenido (GTIN), número de lote, peso neto, cantidad de objetos y fecha de expiración. Dentro de estos parámetros, todos nos interesan y serán gestionados en la base de datos a excepción de las direcciones de remitente y destinatario.

Patrullaje Autónomo en el Almacén

Si se han recibido paquetes, después de escanearlos comenzará esta etapa, sino, esta será la etapa inicial del proceso. El robot comienza un recorrido autónomo (a través de sus sensores integrados, como cámaras y LiDAR) por toda la superficie del almacén, sabiendo en cada momento en qué sección del almacén se encuentra.

Durante este patrullaje, el robot recibe mensajes sobre el estado de los paquetes ya almacenados a través de mensajes transmitidos mediante tecnología *Bluetooth*, concretamente, mensajes de baja energía (*Bluetooth Low Energy*, *BLE*).

Para medir y enviar la información, cada paquete cuenta con un sensor de temperatura y humedad “*SI7021-A20-GM1R*”, y un *beacon* “*WRL-21293*”, ambos componentes desarrollados por la empresa *Sparkfun Electronics*.

De este modo, los datos medidos por el sensor son enviados por el *beacon* vía *BLE*. Estos mensajes, serán recibidos por el robot que estudiará los mensajes enviados y en caso de detectar alguna anomalía, informará al sistema maestro del error producido, en qué sección ha acontecido y las coordenadas exactas dentro del almacén, facilitando la intervención inmediata por parte del personal

logístico, reduciendo el tiempo de respuesta ante posibles incidencias y evitando distribuir elementos que no estén en un estado adecuado.

Retorno a la base de carga

Una vez completado el recorrido de patrullaje, el robot retorna de forma autónoma a su estación base de carga, punto desde el que partió inicialmente. Esta acción garantiza que el robot esté preparado para realizar una nueva ronda de control en cuanto se le ordene, optimizando así su disponibilidad y autonomía operativa.

1.2. Simulación

Dado que el envío de los componentes adicionales necesarios para implementar la funcionalidad completa del proyecto no se producía a tiempo, se optó por desarrollar una simulación completa como alternativa. Para ello, se emplearon las herramientas *Gazebo* y *RViz*, ambas presentes en el framework *ROS*, creando de este modo un entorno virtual preciso que permite probar las funcionalidades implementadas.

Mediante *Gazebo*, se diseñó un entorno que busca reflejar con una alta fidelidad un almacén logístico tipo de una empresa. Este modelo, incluye características propias de un almacén a gran escala como una elevada capacidad de almacenaje y una disposición estructurada por secciones, que a través del uso de robots podría incrementar significativamente el rendimiento operativo.

En la imagen mostrada a continuación, se puede observar dicho entorno de simulación, así como el robot posicionado en él.

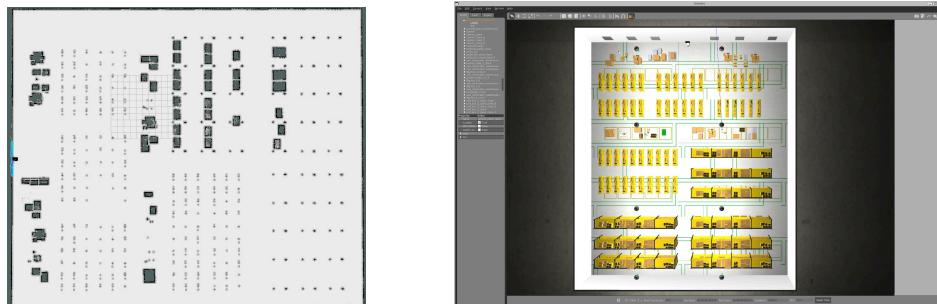


Figura 3. Entorno de simulación y su mapeado.

El entorno está definido en el archivo '*almacen.world*', este contiene la configuración de todos los elementos presentes en la simulación como las diferentes estanterías, zonas de almacenamiento o paquetes recibidos por el centro, junto a sus coordenadas y disposición geométrica.

Por otro lado, como bien se comentó anteriormente, cada paquete cuenta con un sensor de temperatura y humedad, y un beacon. Para simular esto, enviamos paquetes de bytes hexadecimales.

Este entorno, nos va a permitir probar y validar los algoritmos desarrollados, permitiendo evaluar el funcionamiento del robot en condiciones controladas antes de implementarlo en un escenario real. A continuación, se detallarán todos estos aspectos.

2. Implementación de subtareas

El sistema está gobernado mediante una máquina de estado, que organiza y gestiona el flujo de ejecución del programa. Dentro de esta estructura, se definen cuatro tareas principales, que constituyen el núcleo del funcionamiento del sistema.

Tras esto, vamos a explicar en detalle cada uno de los subprocesos ejecutados además del funcionamiento general de la máquina de estados.

2.1. Funcionamiento general de la máquina de estados

Para mantener una estructura de control lo más clara posible se ha implementado todo el desarrollo de las tareas en base a una máquina de estados que permita ejecutar las acciones requeridas en base a dichos estados. De esta manera el funcionamiento de dicha máquina sería el siguiente.

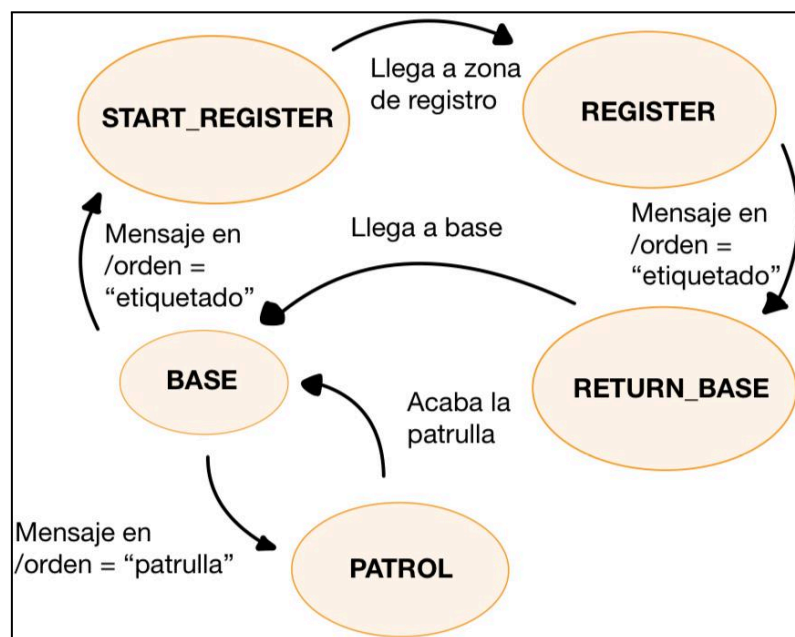


Figura 4. Máquina de estados.

El estado inicial, se corresponde con 'BASE' que podría entenderse como un estado de reposo donde el robot espera la acción que se le pida a través del tópico "/orden".

En el caso de recibir la cadena "etiquetado", ocurre una transición al estado 'START_REGISTER', que manda a través de una acción *move_to* al robot a la zona del registro de nuevos paquetes.

Una vez situado, se manda un mensaje por el topic /caja_ready que activa la señal, la cual simula un sensor de distancia. Cuando se activa esta señal se produce una nueva transición al estado 'REGISTER' encargado de extraer la información de las etiquetas, obteniendo los datos necesarios de cada paquete y registrándolos en la base de datos. Desde este estado se puede dar por finalizado el proceso de registro mediante la publicación de la misma cadena "etiquetado" en el tópico "/orden" de manera que tendría lugar una transición al estado 'RETURN_TO_BASE' que a través de una acción *move_to* redirige el robot a la base.

Si desde 'BASE' se publica en el tópico la cadena "patrulla", tiene lugar una transición al estado 'PATROL', este ejecuta una ruta a lo largo del almacén donde comprueba el estado de cada paquete, tras esto, volverá a la base transicionando de nuevo al estado 'BASE'.

2.2. Lectura de etiquetado

Esta tarea, se encarga de leer las etiquetas de los paquetes recibidos recientemente en el almacén y enviar los datos obtenidos para su gestión en la base de datos. Para llevarlo a cabo, contamos con 2 scripts, '*extract_ocr.py*' y '*ocr_pre_processor.py*', implementando un flujo de trabajo para la extracción de información de etiquetas.

En la implementación, está dividida en dos estados, 'START_REGISTER' se encarga de colocar al robot en la zona de etiquetado mandando una petición al servidor de la acción "move_to" del robot mediante el tópico *"/goal"* y cuando este llega a la zona el estado pasa a 'REGISTER' que se encarga de la propia tarea de escaneo e identificación de la información.

Captura, preprocesado y publicación - *ocr_pre_processor.py*

Este script se encarga de recibir imágenes a través de un tópico, correspondientes a la cámara RGB del robot. Una vez recibida la imagen, esta se convierte a una imagen compatible con *OpenCV*, debido a que esto será lo que utilizaremos para llevar a cabo el preprocesado de la imagen. La etapa de preprocesado, es fundamental debido a que conseguiremos que el algoritmo sea más robusto frente a cambios de iluminación o variaciones respecto al ángulo del que se toman las imágenes.

Este preprocesado, consta de una etapa inicial de recorte de la etiqueta en función del color detectado. Consecutivamente, se convierte la imagen recortada a escala de grises, donde además, se filtra, umbraliza y aplican diversas operaciones morfológicas para optimizar y mejorar la detección de caracteres y legibilidad del texto. Tras esto, se utiliza *Canny* y *Hough* para detectar bordes y líneas, respectivamente. Estas líneas, son estudiadas agrupando y fusionando las cercanas, y ordenarlas según su posición vertical, recortando de nuevo la imagen según estas.

Además, para evitar procesar imágenes duplicadas, se genera un hash *MD5* que permite identificar si la imagen ya ha sido analizada previamente, evitando procesarla de nuevo gestionando eficientemente los recursos.

Si la imagen es nueva, se guarda en el directorio *"/tmp/ocr_input/"* para que '*extract_ocr.py*' pueda procesarla. Una vez que este finaliza la extracción del texto, '*ocr_pre_processor.py*' monitorea los resultados publicados en el archivo *json* del directorio *"/tmp/ocr_output/"*. Finalmente, publica en los tópicos correspondientes de cada campo la información extraída y elimina los datos guardados en los directorios utilizados.

Extracción de texto con OCR - *extract_ocr.py*

Este script es responsable de procesar las imágenes almacenadas en *"/tmp/ocr_input/"*. De estas, extrae la información relevante mediante *Tesseract OCR*.

Inicialmente, monitorea el directorio *"/tmp/ocr_input/"* detectando nuevas imágenes. Cuando se sube una nueva, esta se lee con *Pillow* y su contenido se extrae con *Tesseract*. Tras extraer el texto, analiza y estructura la información obtenida identificando los valores correspondientes a cada campo, en este caso, los de interés son el código SSCC, número de lote, peso, fecha de envío y cantidad de objetos.

Por último, este script se encarga de volcar los datos obtenidos al fichero '*results.json*' en el directorio de salida. Este estado finaliza cuando se ha comprobado que los datos extraídos se han obtenido correctamente, produciéndose una transición de estado y volviendo el robot a la base.

2.3. Vuelta a base

Una vez se hayan registrado todos paquetes finalizará el estado mediante la publicación de la cadena “etiquetado” en el tópico /orden lo que provocará el cambio de estado a ‘RETURN_BASE’ donde se le manda una petición al servidor de la acción move_to para posicionar al robot en la base.

2.4. Patrulla

Este estado es el que fundamenta el funcionamiento global del programa, es el encargado de que el robot patrulle el almacén logístico detectando posibles problemas en los paquetes almacenados. Para implementarla, se deben de tener en cuenta las distintas secciones que componen el almacén y los distintos puntos que conforman la ruta.

Navegación

La ruta de navegación queda definida como un conjunto consecutivo de puntos “waypoints” que contienen los valores de posición y orientación en el espacio tridimensional. Además, estos puntos contienen información acerca de a qué sección pertenecen. Estas secciones representan subdivisiones del almacén organizadas según el tipo de paquetes que contienen, estas están numeradas de 0 a 7.

Además, durante la patrulla, el robot publica actualizaciones del estado actual según la sección en la que se encuentra, calcula y publica el tiempo que pasa en cada sección. Por otro lado, como se ha comentado va enviando los objetivos de navegación. Además, espera la confirmación de llegada a cada waypoint o un tiempo máximo de 120 segundos.

Por último, regresa a la base si completa la patrulla o se reporta un fallo al no lograr llegar a un punto. Esto lo hace ya que el último waypoint se corresponde con la posición de la base.

Simulación de beacons

El objetivo de la patrulla es detectar posibles fallos con los paquetes e informar de estos. Para ello, como se especificó anteriormente, se utilizan sensores y beacons en cada paquete. En caso de que se detecte algún problema en los parámetros medidos de temperatura o humedad, se informará del problema que ha ocurrido a través de un tópico, tras el que se obtendrá la posición y la sección en el momento que ha saltado el error.

La simulación de los Beacons se gestiona desde el nodo ble.py, cuyo objetivo principal es generar paquetes de datos que imitan los valores que un beacon real enviaría por bluetooth si se configura en modo custom, el único de los 3 modos de configuración de los Beacons que permite personalizar el paquete de hasta 31 bytes de tamaño. El paquete diseñado incluye 4 bytes para un identificador único, 2 bytes para una medida de temperatura y otros 2 bytes para una medida de humedad. Por ejemplo, si se genera un paquete con ID 10000001, temperatura 25°C y humedad 50%, el paquete hexadecimal resultante sería algo como 0000001900190032.

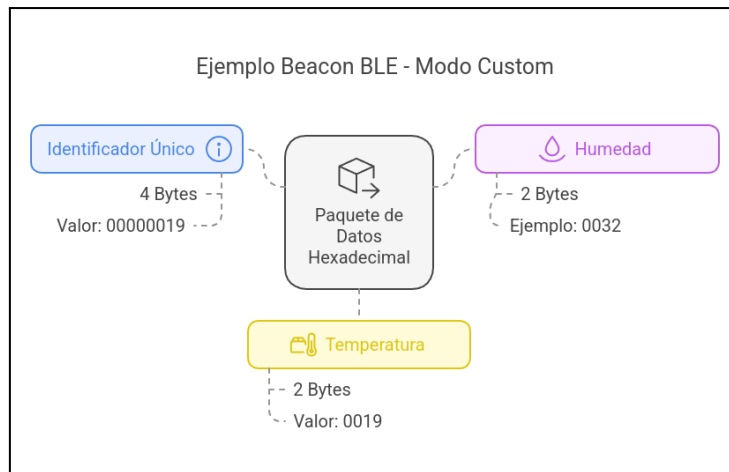


Figura 5. Ejemplo de paquete de datos hexadecimal enviado por un Beacon BLE.

El nodo está diseñado para que empiece a emitir Beacons cuando el modo "patrulla" está activo, tras esto el nodo genera y publica paquetes BLE a intervalos definidos según la sección en la que se encuentra. Para cada sección, se simulan 10 beacons, cada uno con valores únicos de ID, temperatura y humedad generados aleatoriamente entre un intervalo coherente, asegurando que cada simulación sea única.

En resumen, el nodo ble.py actúa como una fuente de datos simulados que permite probar y validar el sistema en condiciones controladas y es esencial para entender cómo poder tratar los beacons BLE en un entorno real.

3. Base de datos

La base de datos es la canal común que enlaza las acciones del robot y su finalidad de servir como sistema de control de inventario, se necesita que sea fluida, con gran capacidad de almacenamiento y a poder ser que ofrezca un entorno visual atractivo para cualquier tipo de consulta que se necesite realizar.

La elaboración de la base de datos se comenzó a elaborar con la librería de python psycogp2 en el entorno local, observadas dificultades y la laboriosa implementación al cambiar de entorno de trabajo durante las distintas probaturas en el desarrollo del proyecto se traspasó a un entorno con servicio gratuito limitado llamado supabase, ofreciendo un entorno visual que permite una elaboración y trabajo manual mucho más sencillo que al tenerla depositada en local, además operar la base de datos con supabase permite hacer uso de su API que cuenta con funciones sencillas e intuitivas para la creación, edición de campos de la base de datos y eliminación de productos.

3.1. Supabase

Supabase es una plataforma que simplifica el desarrollo de aplicaciones al ofrecer una base de datos potente y herramientas para gestionar datos en la nube. Para el desarrollo del proyecto necesitábamos construir un sistema donde diferentes nodos de ROS necesitan consultar y/o modificar información en tiempo real, como datos sobre productos, errores detectados o estados actualizados. Aquí es donde Supabase entra en juego, actuando como el corazón que conecta todo.

Supabase nos proporciona una base de datos centralizada basada en PostgreSQL, donde se almacenan todos los detalles del inventario. Los nodos pueden crear nuevos registros, modificarlos o

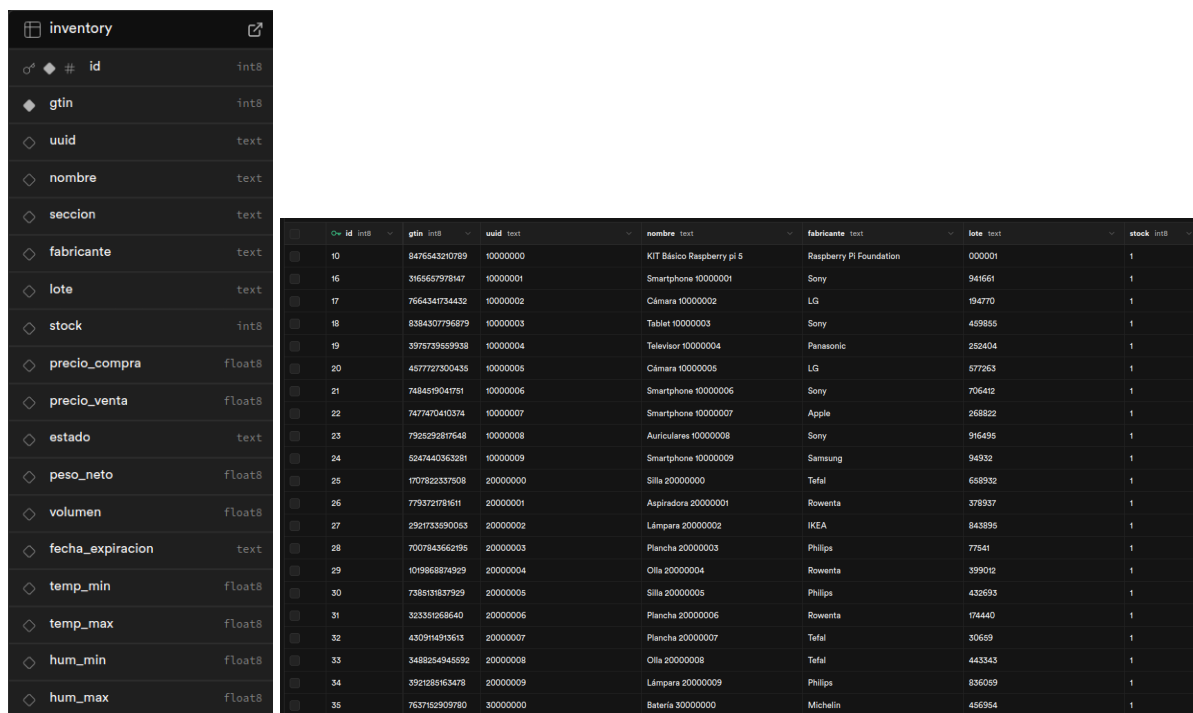
eliminarlos según sea necesario, todo de manera fluida y sin complicaciones. Supabase actúa como un puente que permite a los humanos y al robot colaborar de manera eficiente.

Lo mejor de Supabase es que maneja toda esta infraestructura de datos sorteando configuraciones complejas. Esto nos ha permitido poder centrarnos en lo importante, que los nodos ROS hagan su trabajo de manera eficiente, confiando en que los datos siempre estarán disponibles y actualizados.

El principal inconveniente de usar supabase es que necesitamos un entorno con python superior a la versión 3.8, es decir python3.9 como mínimo. Esto, como para utilizar otros nodos del proyecto es necesario activar un entorno conda, lo cual puede suponer conflictos de versiones de python2 y python3 si no se configura correctamente en el sistema.

3.2 Gestión de Base de Datos

La base de datos cuenta con 18 campos para cada producto que haya en el almacén, entre ellos se encuentran campos como GTIN, lote, peso, o fecha de expiración que se reciben al leer una etiqueta de un nuevo producto que entra en el almacén. Por otro lado cuenta con el campo UUID, que almacena el identificador de Beacon unido al producto, o rangos de temperatura y humedad que nos permitirá llevar un control de seguridad mediante la lectura del paquete de datos que mandan los Beacons.



The image shows the Supabase database interface. On the left, the 'inventory' table schema is displayed with 18 columns: id (int8), gtin (int8), uuid (text), nombre (text), seccion (text), fabricante (text), lote (text), stock (int8), precio_compra (float8), precio_venta (float8), estado (text), peso_netto (float8), volumen (float8), fecha_expiracion (text), temp_min (float8), temp_max (float8), hum_min (float8), and hum_max (float8). On the right, a table view shows 35 records of products with columns: id, gtin, uuid, nombre, fabricante, lote, and stock. The records include various products like Raspberry Pi 5, Smartphones, Cameras, Tablets, Televisors, Auriculares, Sillas, Aspiradores, Lámparas, Planchas, Olla, and Batería.

id	gtin	uuid	nombre	fabricante	lote	stock
10	8476543210789	10000000	KIT Básico Raspberry pi 5	Raspberry Pi Foundation	000001	1
16	365657978947	10000001	Smartphone 10000001	Sony	941661	1
17	7654341734432	10000002	Cámara 10000002	LG	194770	1
18	8384307796879	10000003	Tablet 10000003	Sony	459855	1
19	3975739559938	10000004	Televisor 10000004	Panasonic	252404	1
20	4577727300435	10000005	Cámara 10000005	LG	577263	1
21	7484519047951	10000006	Smartphone 10000006	Sony	705412	1
22	7477470410374	10000007	Smartphone 10000007	Apple	268822	1
23	7925292817548	10000008	Auriculares 10000008	Sony	916495	1
24	5247440363281	10000009	Smartphone 10000009	Samsung	94332	1
25	1707822337508	20000000	Silla 20000000	Tefal	658932	1
26	779372191611	20000001	Aspiradora 20000001	Rowenta	378937	1
27	2921733590053	20000002	Lámpara 20000002	IKEA	843895	1
28	700784562195	20000003	Plancha 20000003	Philips	77541	1
29	101965874929	20000004	Olla 20000004	Rowenta	399012	1
30	7388131837929	20000005	Silla 20000005	Philips	432693	1
31	323391928640	20000006	Plancha 20000006	Rowenta	174440	1
32	430914913613	20000007	Plancha 20000007	Tefal	30659	1
33	3488254945592	20000008	Olla 20000008	Tefal	443343	1
34	3921289163478	20000009	Lámpara 20000009	Philips	836059	1
35	7637152909780	30000000	Batería 30000000	Michelin	456954	1

Figura 6. Base de datos en Supabase.

En etiquetado

El nodo 'inventory_manager.py' es una pieza fundamental pensada para actuar en todo momento además de en el control de registro de productos por lectura de etiqueta. En el sistema de gestión de inventario basado en ROS. Su propósito principal es mantener una conexión constante con la base de datos alojada en Supabase para gestionar productos de manera eficiente y en tiempo real.

El nodo se conecta a Supabase utilizando credenciales específicas, tal y como especifica la API, y escucha varios topics. Estos topics permiten que el nodo interactúe con otros componentes del sistema para llevar a cabo diferentes operaciones:

- Registro de productos: A través del tópico `/inventory_moviles/register`, el nodo recibe datos de nuevos productos y los inserta en la base de datos.
- Actualización de productos: Desde el tópico `/inventory_moviles/update`, se pueden modificar atributos específicos de productos existentes.
- Eliminación de productos: Mediante el tópico `/inventory_moviles/delete`, se pueden eliminar productos identificados por su ID o GTIN.
- Visualización de la base de datos por terminal: Cada segundo, el nodo consulta la base de datos para generar y publicar un resumen del inventario en el tópico `/inventory_moviles/summary` de algunos campos de cada ID.

Adicionalmente, el nodo recibe información dinámica de otros topics relacionados con la lectura de las etiquetas de nuevos productos de entrada. De esas lecturas obtendrá los valores de GTIN, número de lote, cantidad y peso neto, que el nodo utiliza para gestionar automáticamente el inventario.

Uno de los aspectos más interesantes del nodo es que sólo maneja los productos que ya están registrados en la base de datos, por simplificación de funcionalidades en este campo, ya que la introducción de un nuevo GTIN en la base de datos supone informar de muchos más campos que la etiqueta no proporciona. Cuando se detecta un producto cuyo GTIN ya existe, el nodo no simplemente incrementa un campo de stock genérico. En lugar de eso, crea tantas nuevas entradas individuales en la base de datos como lo indica el tópico `/inventory/label/count`. Cada entrada representa una nueva instancia del producto, pero mantiene consistencia al copiar los atributos del producto original.

En la función `add_new_product()`, cuando un GTIN es encontrado en la base de datos:

1. El nodo recupera todos los detalles del producto existente para añadirlos a la nueva entrada.
2. Solo los valores que son específicos del nuevo lote, como el peso neto, el número de lote y la fecha de expiración, se sobrescriben utilizando los datos proporcionados por los topics relevantes.
3. A continuación, el nodo inserta en la base de datos una nueva entrada para cada unidad indicada por el tópico `/inventory/label/count`, manteniendo un `stock` de 1 para cada una.

Este enfoque garantiza que cada unidad de un producto pueda ser rastreada individualmente, tiene como consecuencia que tras la lectura de la etiqueta tendría que haber operarios de almacén que se encargaran de individualizar cada lote de entrada al almacén para que hubiera concordancia con la base de datos.

En patrulla

El nodo `read_ble.py` es un componente diseñado para procesar y validar datos transmitidos por Beacons BLE simulados en el sistema, aunque serviría para procesar paquetes de datos reales de Beacons con la configuración precisa. Este nodo combina la capacidad de comunicación en tiempo real con la base de datos, permitiendo verificar y gestionar información crítica sobre productos en un inventario.

El nodo tiene como propósito principal escuchar datos transmitidos por beacons a través del tópico `/ble/message` en todo momento, aunque por como está pensada la elaboración de los Beacons simulados, sólo podrá, durante la patrulla, procesar y verificar su validez comparándolos con la información almacenada en la base de datos. Los datos incluyen un identificador único del beacon, así como valores de temperatura y humedad. Basándose en esta información, el nodo realiza

comprobaciones para detectar discrepancias, como valores fuera de los rangos esperados o una asignación incorrecta a una sección del inventario.

En caso de detectar errores, el nodo toma dos acciones principales:

1. Actualizar el estado del producto: Modifica el campo "estado" del producto en la base de datos, registrando el tipo de error detectado.
2. Generar marcadores visuales: Crea marcadores en RViz en la posición actual del robot, lo que permite identificar visualmente los productos problemáticos.

Cuando el nodo recibe un mensaje de beacon en formato hexadecimal, este lo desempaqueta como un conjunto de tres valores:

ID: Representado por los primeros 4 bytes del paquete.

Temperatura: Codificada en los 2 bytes siguientes.

Humedad: Contenida en los últimos 2 bytes.

A continuación, consulta la base de datos en busca de información del producto asociado al beacon, verifica si:

- La temperatura y la humedad están dentro de los rangos permitidos definidos en la base de datos con valores mínimos y máximos.
- El producto está asignado a la sección correcta, basada en el primer valor del identificador del beacon.
- En caso de que el producto no exista en la base de datos, lo marca como "Producto no encontrado".

Si se detecta algún error de los anteriores, el nodo actualiza el campo "estado" del producto en la base de datos para reflejar el problema específico, como "Fuera de sección", "Temperatura fuera de rango" o "Humedad fuera de rango". Esto permite mantener un registro claro de los problemas identificados.

Para facilitar la localización de productos con errores, el nodo genera marcadores visuales, representados como esferas de color rojo, en RViz. Estos marcadores se posicionan en la ubicación actual del robot, basada en los datos de odometría recibidos del tópico `/robot/robotnik_base_control/odom``. Además, Incluyen un identificador único, basado en la marca de tiempo, y se registran, junto con el ID de la base de datos del producto erróneo, en un archivo JSON local para eliminar el marcador cuando el error se arregle.

3.3 Interacción Manual

El nodo ``use_database.py`` actúa como una interfaz interactiva que permite gestionar el inventario de forma sencilla desde el terminal. Ofrece opciones para registrar, modificar, eliminar productos y corregir errores detectados.

Al iniciar, el nodo establece una conexión con Supabase carga un archivo local llamado ``productos_erroneos.json``, que contiene información sobre productos que presentaron errores y han sido marcados visualmente en RViz por su uso simultáneo del nodo ``read_ble.py``.

El funcionamiento del nodo gira en torno a un menú interactivo que permite realizar diversas acciones. Si el usuario desea registrar un producto, el nodo recopila los datos necesarios y publica un mensaje en el tópico correspondiente para que ``inventory_manager.py``, procese el registro y lo almacene en la base de datos. De manera similar, si se desea modificar un producto, el nodo solicita

el campo a actualizar y publica un mensaje con los datos modificados. La eliminación de productos también es gestionada de forma sencilla, permitiendo al usuario identificar el producto a eliminar mediante su ID o GTIN.

Una de las funcionalidades más destacadas del nodo es la corrección de errores. Esto se integra con el trabajo del nodo ``read_ble.py``, que detecta discrepancias en los datos de los beacons BLE y genera marcadores visuales en RViz para destacar distintos errores. ``use_database.py`` permite al usuario ver una lista de estos productos con errores, seleccionarlos, y confirmar su corrección o no. Tras la confirmación, elimina el marcador correspondiente en RViz, actualiza el estado del producto en la base de datos a "Correcto" y lo elimina del archivo ``productos_erroneos.json``, asegurando que los datos del sistema estén sincronizados.

4. Vídeo de los resultados obtenidos

Vídeo: <https://youtu.be/PkZ8wWkjVw4>

5. Bibliografía

https://github.com/ggari-robotnik/summit_xl_sim - Robotnik

<https://github.com/aws-robotics/aws-robomaker-small-warehouse-world>

<https://www.digikey.es/es/products/detail/silicon-labs/SI7021-A20-GM1R/5048935?srsId=AfmBOoqmWIVTZHC0mikBF3rt94aMXEYYxJTY4NI7z12rDpccLEw6P6x> - Sensor

<https://www.digikey.es/es/products/detail/sparkfun-electronics/WRL-21293/18635166?s=N4lgTCBcDaIOoCUAyBaMBGMBOAzCAugL5A> - Beacon

<https://robotnik.eu/wp-content/uploads/2021/06/Robotnik-SUMMIT-XL-Datasheet-210628-ES.pdf>

<https://supabase.com/docs/reference/python/is> - API Python Supabase

<https://github.com/supabase/supabase-py> - Github Supabase

<https://www.postgresql.org/docs/current/index.html> - PostgreSQL