



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Escola Tècnica
Superior d'Enginyeria
Informàtica



etsinf

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

7 de junio de 2024

MANUAL DE PROGRAMACIÓN PREPARACIÓN DE PEDIDOS

PROYECTO RII 2 - PROGRAMACIÓN DE ROBOTS | PR2-A04
Grado en Informática Industrial y Robótica

Autores/Autoras:

Marcos Belda Martínez | mbelmar@etsinf.upv.es

Ángela Espert Cornejo | aespcor@etsinf.upv.es

Lourdes Francés Llimerá | lfralli@epsa.upv.es

Carla Hidalgo Aroca | chidaro@etsii.upv.es

Tutores/Tutoras:

Juan Francisco Blanes Noguera | pblanes@ai2.upv.es

Joan Josep Fons Cors | jfons@dsic.upv.es

Marina Vallés Miquel | mvalles@isa.upv.es

Eduardo Vendrell Vidal | even@upv.es

RESUMEN

En este documento se explica brevemente el desarrollo *software* aportado junto a la [Propuesta de Automatización](#). En primer lugar, se esclarecen las relaciones de interacción entre los componentes aportados. Posteriormente, se explican las funciones principales y la estructura organizativa de cada una de las soluciones. Para terminar, se concluye el documento aportado a una reflexión y una síntesis del trabajo que se realizaría de cara a posteriores fases del proyecto.

Palabras clave: actuador, BBDD, C/C++, ESP32, MQTT, programación avanzada, Python, sensor.

ABSTRACT

This document briefly explains the software development provided along with the [Automation Proposal](#). First, the interaction relationships between the components provided are clarified. Then, the main functions and the organizational structure of each of the solutions are explained. Finally, the document is concluded with a reflection and a summary of the work that would be carried out for later phases of the project.

Key words: actuator, advanced programming, C/C++, databases, ESP32, MQTT, Python, sensor.



TABLA DE CONTENIDOS

1. INTRODUCCIÓN	3
ASPECTOS A TENER EN CUENTA	3
SOLUCIONES DESARROLLADAS	3
2. RELACIÓN ENTRE LAS SOLUCIONES	3
3. ORGANIZACIÓN DE LAS SOLUCIONES	5
<i>ESP32-01. SENSORA/ACTUADORA</i>	5
LENGUAJE DE PROGRAMACIÓN Y DEPENDENCIAS REQUERIDAS	5
FICHEROS Y FUNCIONES PRINCIPALES	5
CÓMO EJECUTAR LA SOLUCIÓN	13
<i>ESP32-02. CONTROLADORA</i>	14
LENGUAJE DE PROGRAMACIÓN Y DEPENDENCIAS REQUERIDAS	14
FICHEROS Y FUNCIONES PRINCIPALES	14
CÓMO EJECUTAR LA SOLUCIÓN	19
<i>ROBODK. SCRIPTS DE PYTHON</i>	19
LENGUAJE DE PROGRAMACIÓN Y DEPENDENCIAS REQUERIDAS	19
FICHEROS, IMPORTACIONES NECESARIAS Y FUNCIONES PRINCIPALES	20
CÓMO EJECUTAR LA SOLUCIÓN	25
<i>ALGORITMO DE COLOCACIÓN DE DISPOSITIVOS (C++)</i>	25
LENGUAJE DE PROGRAMACIÓN Y DEPENDENCIAS REQUERIDAS	25
FICHEROS Y FUNCIONES PRINCIPALES	26
CÓMO EJECUTAR LA SOLUCIÓN	35
4. CONCLUSIONES Y TRABAJO FUTURO	35
5. BIBLIOGRAFÍA	36
APLICACIONES EMPLEADAS	36
DOCUMENTOS DE REFERENCIA	36

1. INTRODUCCIÓN

ASPECTOS A TENER EN CUENTA

El desarrollo *software* que se explica en el presente documento, forma parte de una versión preliminar de lo que sería la solución final. No se puede afirmar que las soluciones funcionen correctamente en todos los casos, sería durante las siguientes fases del proyecto donde se depurarían los posibles problemas y se llevarían a cabo las mejoras pertinentes.

SOLUCIONES DESARROLLADAS

El *software* aportado consiste en la programación de dos dispositivos embebidos tipo ESP32-S3, un conjunto de *scripts* en Python para la simulación en RoboDK, y por último, el desarrollo de un algoritmo para calcular la colocación de los dispositivos dentro de las cajas.

PROGRAMACIÓN DE LAS ESP32-S3

La primera de ellas se trata de la ESP32-01, esta se encarga de las tareas de sensorización y de la gestión de los actuadores. El segundo dispositivo es la ESP32-02, este se encarga de las tareas control, ya que toma decisiones en base a la información que recibe de la otra ESP32, enviando las órdenes oportunas al resto de dispositivos de la estación. Ambas ESP32 implementan mecanismos de comunicación indirecta (vía MQTT).

SCRIPTS DE PYTHON

Estos *scripts* permiten comunicar los elementos de la simulación con el mundo real vía MQTT.

ALGORITMO DE COLOCACIÓN DE DESPOSITIVOS

Se trata de un algoritmo simplificado al que se aporta una lista de dispositivos que se desean introducir en una caja, la lista se procesa para conseguir una secuencia de llenado que sería almacenada en la base de datos hasta que el proceso requiera de esa información.

2. RELACIÓN ENTRE LAS SOLUCIONES

La unión de todas las soluciones es clave para el correcto funcionamiento del proceso final. La finalidad de este apartado es aportar una mejor comprensión de las relaciones entre las soluciones; para ello, a continuación se muestran las Figuras 3.1 y 3.2, las cuales consisten en diagramas explicativos.

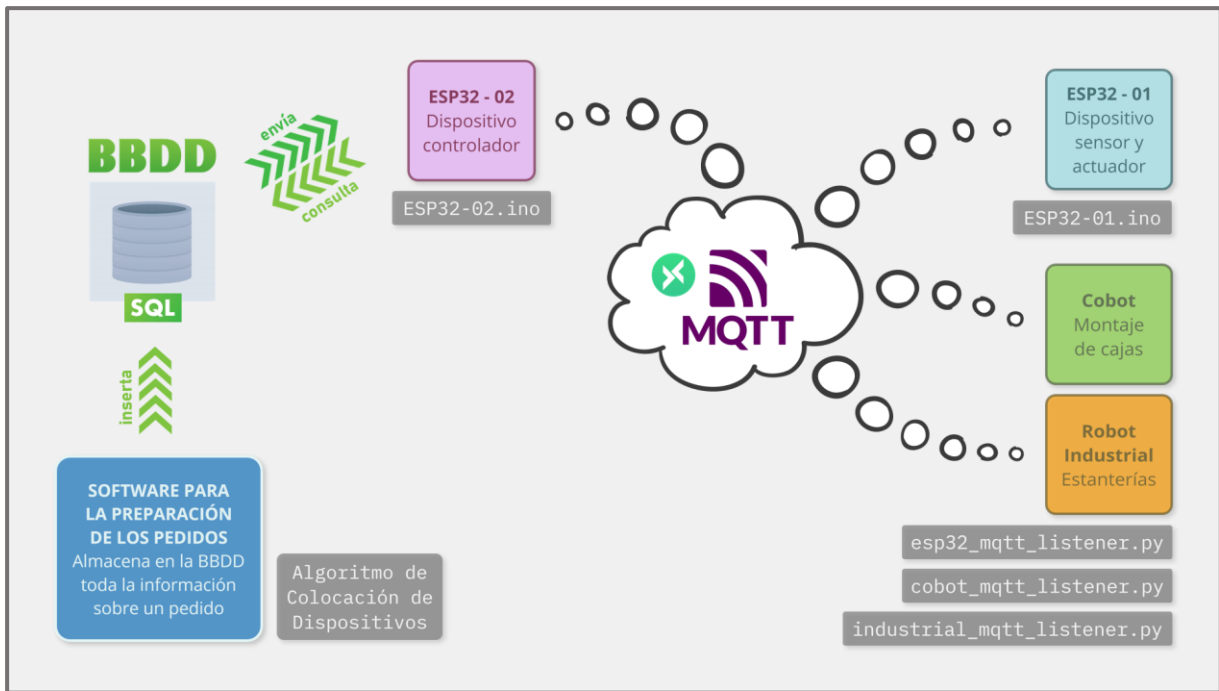


Figura 2.1: Diagrama que muestra las soluciones (recuadros grises) junto a los elementos que las ejecutan.

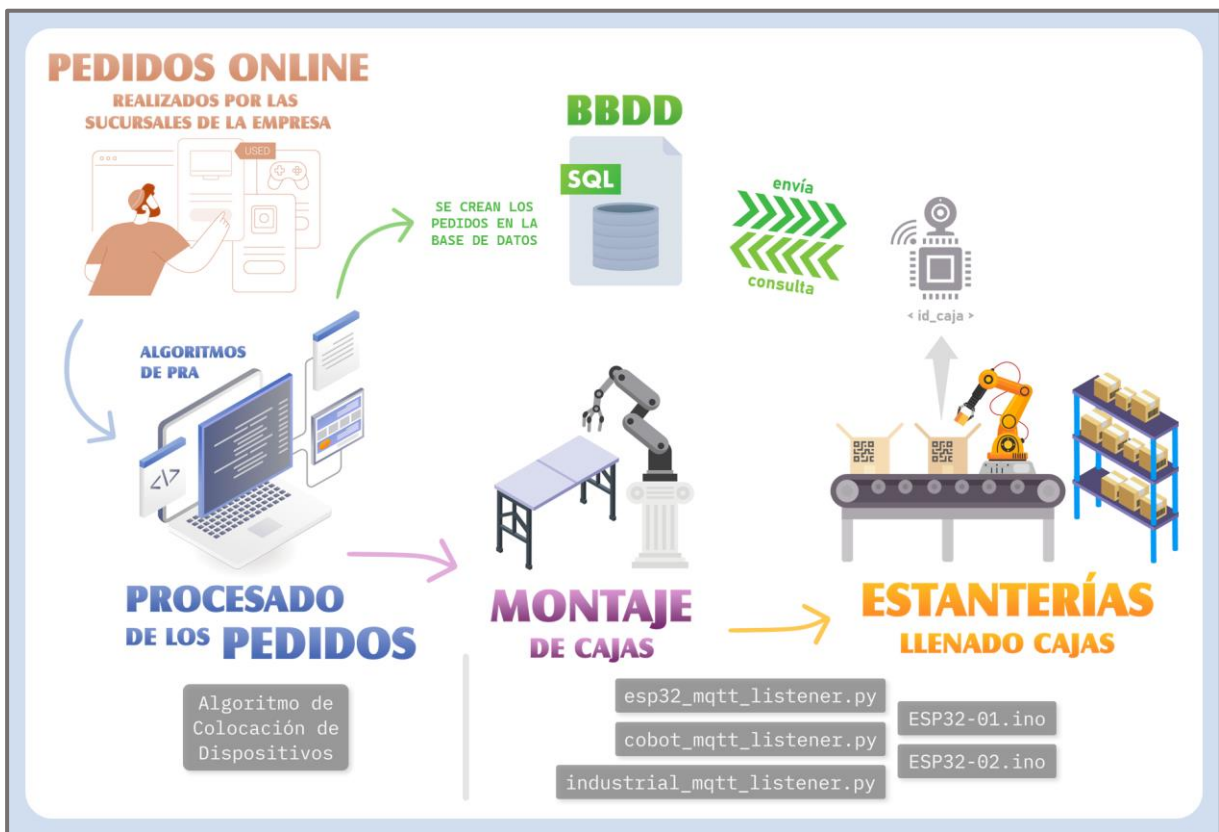


Figura 2.2: Diagrama que muestra a que fase del proceso pertenece cada solución.

3. ORGANIZACIÓN DE LAS SOLUCIONES

ESP32-01. SENSORA/ACTUADORA

LENGUAJE DE PROGRAMACIÓN Y DEPENDENCIAS REQUERIDAS

El lenguaje de programación utilizado para el desarrollo de esta solución es C++, a través del entorno [Arduino IDE 2.0](#). Para que el código de esta primera ESP32 funcione correctamente, es necesario tener instalado en el entorno el administrador de tarjetas ESP32, ciertas librerías disponibles en Arduino y algunos ficheros desarrollados por otros usuarios.

Administrador de tarjetas de Arduino

- [esp32](#) por *Espressif Systems* (versión 2.0.17).

Administrador de librerías de Arduino

- [ArduinoJson](#) por *Benoit Blanchon* (versión 7.0.4).
- [Bounce2](#) por *Thomas O Fredericks* (versión 2.71).
- [EventButton](#) por *Philip Fletcher* (versión 1.0.3).
- [PubSubClient](#) por *Nick O'Leary* (versión 2.8).

Ficheros de desarrollo ajeno

- [ArduinoJson.h](#) por *Benoit Blanchon*.
- [esp_camera.h](#) por *Espressif Systems*.
- [PubSubClient.h](#) por *Nick O'Leary*.
- [quirc.h](#) por *Daniel Beer*.
- [WiFi.h](#) por *Espressif Systems*.
- [WiFiClientSecure.h](#) por *Espressif Systems*.

Ficheros de desarrollo propio

- [config.h](#) por *Grupo PR2-A04* (versión ESP32-01).
- [Proyecto ESP32_02.ino](#) por *Grupo PR2-A04* (basado en el proyecto *ESP32-S3-IoT-Device.ino* de [Joan Josep Fons Cors](#)).

FICHEROS Y FUNCIONES PRINCIPALES

El entorno Arduino IDE 2.0 permite la división en ficheros de un proyecto de Arduino, con esta funcionalidad se consigue un código más cómodo de organizar y legible. El proyecto de Arduino está formado por nueve ficheros y a la hora de construirlo se realiza en orden alfabético.

A continuación, se muestra una breve descripción de los ficheros y de sus funciones, el orden en el que se mencionan es en el que se realiza la construcción del proyecto. Para más información sobre el código se sugiere que se visite el [Repositorio de la ESP32-01 \(GitHub\)](#).

confih.h

```
/**
 * @file    c_config.h
 *
 * @brief   Donde se crean las definiciones, tipos
 *          enumerados y estructuras necesarias.
 *
 * @version 0.1 (versión ESP32-01)
 * @authors Joan Fons' <jjfons@dsic.upv.es>, Grupo PR2-A04'
 * @date    Junio, 2024
 * @section PR2-GIIROB
 */
```

```
//-----[ LIBRERIAS NECESARIAS ]-----//
...
//-----[ COMM BAUDS ]-----//
...
//-----[ IDENTIFICADOR DEL DISPOSITIVO ]-----//
...
//-----[ WIFI ]-----//
...
//-----[ MQTT ]-----//
...
//-----[ SENSORES DE PRESENCIA ]-----//
...
//-----[ CÁMARA PARA LECTURA DE QR ]-----//
...
//-----[ PULSADOR DE EMERGENCIA ]-----//
...
//-----[ RETENEDOR (SERVOMOTOR) ]-----//
...
//-----[ MOTORES DE LAS (MOTORES CC) ]-----//
...
```

c_logger.ino

```
/**
 * @file    c_logger.ino
 * @brief   Definición e implementación de las funciones de logging.
 *
 * En este fichero se definen hasta seis niveles diferentes para realizar
 * la depuración del código por la terminal Serial Monitor. Para cada nivel
 * hay disponibles dos versiones, imprimir un mensaje pasando a la línea
 * siguiente o manteniéndose en la misma.
 */
```

ESP32_01.ino

```
/**
 * @file      ESP32-01.ino
 *
 * @brief     Solución de la ESP32-01 basada en la 'Plantilla Dispositivo IoT'.
 *
 * En este fichero se definen las funciones principales de Arduino, el setup(),
 * donde se realizan las inicializaciones, y el loop(), función que se ejecuta
 * de forma indefinida.
 *
 * @version  0.8   (2024/06/05) Versión para la entrega del día 6 de junio
 * @version  0.7   (2024/04/21) Aplicando estándar de programación BARR-C
 * @version  0.6   (2024/02/21) JSON Management
 * @version  0.5.1 (2024/02/20) SSL fork
 * @version  0.5   (2024/02/19) Mejora en comunicaciones y reconexiones
 * @version  0.4   (2024/02/15) Reestructuración y simplificación
 * @version  0.3   (2024/02/14) Añadido funciones de log
 * @version  0.2   (2023/12/28) Dividido en sub-ficheros
 * @version  0.1   (2023/11/29) Prototipo Inicial Funcional
 *
 * @authors   Joan Fons' <jjfons@dsic.upv.es>, Grupo PR2-A04'
 * @date      Junio, 2024
 * @section   PR2-GIIROB
 */
```

```
/*
 * @brief     Este setup configura conceptos 'core', inicializa la wifi y la
 *           conexión con el bróker MQTT, y ejecuta los siguientes métodos:
 *           - suscribirseATopics() -> topics MQTT a suscribir para recibir
 *           mensajes (h_comunicaciones.ino)
 *           - on_setup()          -> configuración de los pines, inicializa-
 *           ción de variables, etc. (s_setup.ino)
 * @param     void
 * @return    void
 */
void setup(void)
{ ... } /* setup() */
```

```
/*
 * @brief     Este loop ejecuta los siguientes métodos en bucle:
 *           - wifi_loop() -> función bucle WiFi (d_wifi_lib_no_tocar.ino)
 *           - mqtt_loop() -> función bucle MQTT (e_mqtt_lib_no_tocar.ino)
 *           - on_loop()   -> tareas a realizar dentro del 'loop' (w_loop.ino)
 * @param     void
 * @return    void
 */
```



```
void loop(void)
{ ... } /* loop() */
```

d_wifi_lib_no_tocar.ino

```
/**
 * @file d_wifi_lib_no_tocar.ino
 * @brief Definición e implementación de funciones de biblioteca WiFi.
 */

/*****
 *!
 * @brief Función bucle WiFi.
 * @param void
 * @return void
 */
void wifi_loop(void)
{ ... } /* wifi_loop() */

/*****
 *!
 * @brief Función de conexión WiFi.
 * @param void
 * @return void
 */
void wifi_connect(void)
{ ... } /* wifi_connect() */

/*****
 *!
 * @brief Función de reconexión WiFi.
 * @param retries (uint)
 * @return void
 */
void wifi_reconnect(uint retries)
{ ... } /* wifi_reconnect() */
```

e_mqtt_lib_no_tocar.ino

```
/**
 * @file e_mqtt_lib_no_tocar.ino
 * @brief Definición e implementación de las funciones de MQTT.
 */

/*****
 *!
 * @brief Función bucle MQTT.
 * @param void
 * @return void
 */
void mqtt_loop(void)
```

```

{ ... } /* mqtt_loop() */

/*****
/*!
 * @brief Función de conexión MQTT. Se configura el cliente MQTT y se
 *         configura 'mqttCallback' como la función que se invocará al
 *         recibir datos por las suscripciones realizadas.
 * @param clientID (String)
 * @return void
 */
void mqtt_connect(String clientID)
{ ... } /* mqtt_connect() */

/*****
/*!
 * @brief Función de reconexión MQTT.
 * @param retries (int)
 * @return void
 */
void mqtt_reconnect(int retries)
{ ... } /* mqtt_reconnect() */

/*****
/*!
 * @brief Función callback MQTT. Función que se invocará automáticamente al
 *         recibir datos por algún topic sobre el que nos hayamos suscrito.
 * @param topic (char *)
 * @param message (byte *)
 * @param length (unsigned int)
 * @return void
 */
void mqttCallback(char * topic, byte * message, unsigned int length)
{ ... } /* mqttCallback() */

/*****
/*!
 * @brief Función de publicación de mensajes.
 * @param topic (const char *)
 * @param outgoingMessage (String)
 * @return void
 */
void mqtt_publish(const char * topic, String outgoingMessage)
{ ... } /* mqtt_publish() */

/*****
/*!
 * @brief Función de suscripción a los topics.
 * @param topic (const char *)
 * @return void
 */
void mqtt_subscribe(const char * topic)

```

```
{ ... } /* mqtt_subscribe() */
```

f_funciones.ino

```
/**
 * @file f_funciones.ino
 * @brief Definición e implementación de las funciones
 * de inicialización y gestión del hardware.
 */

/*****/
/#!
 * @brief Gestor de interrupción para el pulsador de emergencia. Esta función
 * se aloja en la memoria RAM (hay que usar la directiva de IRAM_ATTR).
 * @param void
 * @return void
 */
void IRAM_ATTR isr(void)
{ ... } /* isr() */

/*****/
/#!
 * @brief Función para conocer la distancia detectada por el ultrasonido.
 * @param sensor Indica qué sensor de la cinta se desea
 * utilizar para realizar la medición.
 * @return Devuelve la distancia en centímetros en formato float.
 */
float getUsDistance(tipo_sensor_t sensor)
{ ... } /* getUsDistance() */

/*****/
/#!
 * @brief Función para inicializar la cámara de la ESP32.
 * @param void
 * @return void
 */
void camera_init(void)
{ ... } /* camera_init() */

/*****/
/#!
 * @brief Función para activar la cámara de la ESP32 para leer un código QR.
 * @param void
 * @return Devuelve el contenido del código QR en formato String.
 */
String camera_get_QR(void)
{ ... } /* camera_get_QR() */

/*****/
/#!
 * @brief Función para inicializar el retenedor (servomotor en la demo).
```

```

* @param int Es el pin al que se quiere conectar en la placa.
* @return void
*/
void retenedor_init(int pin)
{ ... } /* servo_set_pin() */

/*****
/*!
* @brief Función para accionar el retenedor (servomotor en la demo).
* @param accion_retenedor Posición en la que se quiere colocar el retenedor.
* @return void
*/
void retenedor_set_accion(accion_retenedor_t accion_retenedor)
{ ... } /* servo_set_angle() */

```

g_tareas.ino

```

/**
* @file g_tareas.ino
* @brief Definición e implementación de las tareas de sensorización.
*/

/*****
/*!
* @brief Tarea que se encarga de sensorizar el inicio de la cinta 1.
* @param void * parameter -> puntero a la estructura 'estado_cinta'
* @return void
*/
void sensor_inicio_cinta_task(void * parameter)
{ ... } /* sensor_inicio_cinta_task() */

/*****
/*!
* @brief Tarea que se encarga de sensorizar el final de la cinta 1.
* @param void * parameter -> puntero a la estructura 'estado_cinta'
* @return void
*/
void sensor_final_cinta_task(void * parameter)
{ ... } /* sensor_final_cinta_task() */

/*****
/*!
* @brief Tarea que se encarga realizar la lectura del QR
* cuando hay una caja disponible para ser escaneada.
* @param void * parameter -> puntero a la estructura 'estado_cinta'
* @return void
*/
void lectorQR_task(void * parameter)
{ ... } /* lectorQR_task() */

```

h_comunicaciones.ino

```
/**
 * @file h_comunicaciones.ino
 * @brief Definición e implementación de las funciones de comunicación.
 */

/*****
 *!
 * @brief Funcion de suscripción a los topics.
 * @param void
 * @return void
 */
void suscribirseATopics(void)
{ ... } /* suscribirseATopics() */

/*****
 *!
 * @brief Función que gestiona los mensajes
 * recibidos por los diferentes topics.
 * @param topic (char *)
 * @param incomingMessage (String)
 * @return void
 */
void alRecibirMensajePorTopic(char * topic, String incomingMessage)
{ ... } /* alRecibirMensajePorTopic() */

/*****
 *!
 * @brief Función para enviar un mensaje por un topic.
 * @param topic (const char *)
 * @param outgoingMessage (String)
 * @return void
 */
void enviarMensajePorTopic(const char * topic, String outgoingMessage)
{ ... } /* enviarMensajePorTopic() */
```

s_setup.ino

```
/**
 * @file s_setup.ino
 * @brief Definición e implementación de la función de inicialización.
 */

/*****
 *!
 * @brief Función de inicialización de los sensores de presencia, la cámara,
 * el pulsador de emergencia, el retenedor, y los motores de las cintas.
 * En esta función además se crean las tareas de (g_tareas.ino).
 * - sensor_inicio_cinta_task()
 * - sensor_final_cinta_task()
```

```

*           - lectorQR_task()
* @param void
* @return void
*/
void on_setup(void)
{ ... } /* on_setup() */

```

w_loop.ino

```

/**
 * @file w_loop.ino
 * @brief Definición e implementación de la función de bucle.
 */

/*****
 *!
 * @brief Cuando el pulsador de emergencia se presiona, esta
 * función publica vía MQTT un aviso de emergencia.
 * @param void
 * @return void
 */
void on_loop(void)
{ ... } /* on_loop() */

```

CÓMO EJECUTAR LA SOLUCIÓN

Para ejecutar la solución es necesario tener instalado el entorno de Arduino IDE 2.0, además de tener acceso a las [dependencias requeridas](#) mencionadas anteriormente. Por supuesto, también es necesario tener un dispositivo ESP32-S3 ya que, para otros modelos de la misma marca la configuración de los pines podría cambiar.

Una vez se tengan las herramientas *software* y la placa ESP32-S3, el siguiente paso es conectar el *hardware* a los pines correspondientes y alimentar la placa mediante alimentación externa. Finalmente subir el programa a la placa mediante un cable USB, conectando la placa al ordenador donde se encuentre el proyecto de Arduino.

Para subir el programa a la placa hay que seleccionar el dispositivo indicado y establecer la configuración indicada en la Figura 3.1.

Asegurarse de que hay una red Wifi disponible.

Para la demo se establece conexión con la red privada del laboratorio 1B 0.0 (Ángela Ruíz Robles), para establecer conexión con otra red, hay que modificar los parámetros en el fichero `config.h`.

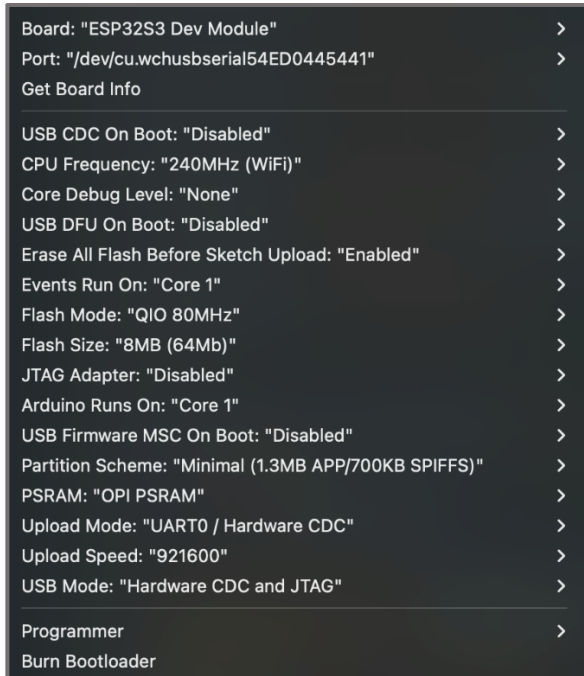


Figura 3.1: Configuración para la pestaña de herramientas.

ESP32-02. CONTROLADORA

LENGUAJE DE PROGRAMACIÓN Y DEPENDENCIAS REQUERIDAS

Al igual que la anterior ESP32, el lenguaje de programación utilizado para el desarrollo de esta solución es C++, a través del entorno Arduino IDE 2.0. Para que el código de esta segunda ESP32 funcione correctamente, es necesario tener instalado en el entorno el administrador de tarjetas ESP32, ciertas librerías disponibles en Arduino y algunos ficheros desarrollados por otros usuarios.

Administrador de tarjetas de Arduino

- [esp32](#) por *Espressif Systems* (versión 2.0.17).

Administrador de librerías de Arduino

- [ArduinoJson](#) por *Benoit Blanchon* (versión 7.0.4).
- [PubSubClient](#) por *Nick O'Leary* (versión 2.8).

Ficheros de desarrollo ajeno

- **ArduinoJson.h** por *Benoit Blanchon*.
- **PubSubClient.h** por *Nick O'Leary*.
- **WiFi.h** por *Espressif Systems*.
- **WiFiClientSecure.h** por *Espressif Systems*.

Ficheros de desarrollo propio

- **buffer_circular.h** por *Grupo PR2-A04* (basado en el proyecto *ejem_buffer_tareas.ino* de [Juan Francisco Blanes Noguera](#)).
- **config.h** por *Grupo PR2-A04* (versión ESP32-02).
- **Proyecto ESP32_02.ino** por *Grupo PR2-A04* (basado en el proyecto *ESP32-S3-IoT-Device.ino* de [Joan Josep Fons Cors](#)).

FICHEROS Y FUNCIONES PRINCIPALES

Al igual que en la anterior solución, se ha dividido el proyecto en ficheros, para conseguir un código más cómodo de organizar y legible. Hay algunos ficheros que se repiten de la anterior solución por lo que no serán explicados, simplemente se hará referencia a ellos. Para más información sobre el código se sugiere que se visite el [Repositorio de la ESP32-02 \(GitHub\)](#).

buffer_circular.h

```
/**
 * @file    buffer_circular_t.h
 *
 * @brief   Definición e implementación de la clase templatizada Buffer Circular.
 *
 * @authors Grupo PR2-A04' <mbelmar@etsinf.upv.es>
 * @date    Junio, 2024
 * @section PR2-GIIROB
```

```

*/
/*****
/!
* @brief Método constructor del Buffer Circular.
* @param void
*/
buffer_circular_t(void)
{ ... } /* buffer_circular_t() */
/*****
/!
* @brief Método destructor del Buffer Circular.
* @param void
*/
~buffer_circular_t(void)
{ ... } /* ~buffer_circular_t */
/*****
/!
* @brief Método para saber si el Buffer Circular está vacío.
* @param void
* @return True or False.
*/
bool isEmpty(void)
{ ... } /* isEmpty() */
/*****
/!
* @brief Método para saber si el Buffer Circular está lleno.
* @param void
* @return True or False.
*/
bool isFull(void)
{ ... } /* isFull() */
/*****
/!
* @brief Método para obtener un elemento del Buffer Circular.
* @param T Puntero a la variable donde se quiere
* guardar el valor del elemento del Buffer.
* @return Devuelve 'buffer_vacio' si no cabe ningún elemento
* más, si no devuelve el elemento obtenido.
*/
err_buffer_t get_item(T * X)
{ ... } /* get_item() */
/*****
/!
* @brief Método para introducir un elemento en el Buffer Circular.
* @param T Elemento que se quiere guardar en el Buffer.
* @return Devuelve 'buffer_lleno' si no cabe ningún elemento más.

```



```

*/
err_buffer_t put_item(T X)
{ ... } /* put_item() */

/*****/
/*!
 * @brief Método para saber cuántos elementos tiene el Buffer Circular.
 * @param void
 * @return Devuelve el número de elementos.
 */
int number(void)
{ ... } /* size() */

/*****/
/*!
 * @brief Método para saber cuántos elementos puede almacenar el Buffer Circular.
 * @param void
 * @return Devuelve el tamaño del Buffer Circular.
 */
int size(void)
{ ... } /* size() */

```

config.h

```

/**
 * @file c_config.h
 *
 * @brief Donde se crean las definiciones, tipos
 * enumerados y estructuras necesarias.
 *
 * @version 0.2 (versión ESP32-02)
 * @authors Joan Fons' <jjfons@dsic.upv.es>, Grupo PR2-A04'
 * @date Junio, 2024
 * @section PR2-GIIROB
 */

//-----[ LIBRERIAS NECESARIAS ]-----//
...
//-----[ COMM BAUDS ]-----//
...
//-----[ IDENTIFICADOR DEL DISPOSITIVO ]-----//
...
//-----[ WIFI ]-----//
...
//-----[ MQTT ]-----//
...
//-----[ VARIABLES PARA LA GESTIÓN DE LAS CINTAS ]-----//
...

```

c_logger.ino

El contenido de este fichero no difiere con el [c_logger.ino de la ESP32-01](#).

ESP32_02.ino

El contenido de este fichero no difiere con el [ESP32_01.ino de la ESP32-01](#).

d_wifi_lib_no_tocar.ino

El contenido de este fichero no difiere con el [d_wifi_lib_no_tocar.ino de la ESP32-01](#).

e_mqtt_lib_no_tocar.ino

El contenido de este fichero no difiere con el [e_mqtt_lib_no_tocar.ino de la ESP32-01](#).

f_funciones.ino

```
/**
 * @file f_funciones.ino
 * @brief Definición e implementación de las funciones
 * de comunicación con la base de datos.
 */

/*****
/*!
 * @brief Función de consulta de la Base de Datos (sin implementar).
 * @param id_caja Identificador de la caja para buscar en la base de datos.
 * @param orden_id_caja Puntero a un documento JSON donde se insertará toda
 * la información necesaria para la orden del industrial.
 * @return void
 */
void consultar_BBDD_y_generar_orden(char * id_caja, JsonDocument * orden_id_caja)
{ ... } /* consulta_BBDD_y_generar_orden () */
```

g_tareas.ino

```
/**
 * @file g_tareas.ino
 * @brief Definición e implementación de tareas de gestión de decisiones.
 */

// Creando un manejadores de las tareas.
//
TaskHandle_t gestion_cintas_task_handle;
TaskHandle_t retenedor_camara_task_handle;

/*****
/*!
 * @brief Tarea que se encarga analizar la lectura del QR. De esta manera se
 * consigue retener la caja correctamente dependiendo de su tamaño y se
 * genera/envía la orden para el robot industrial.
```

```

* @param void * parameter -> puntero a la estructura 'estado_cinta'
* @return void
*/
void retenedor_camara_task(void * parameter)
{ ... } /* retenedor_camara_task() */

/*****
/#!
* @brief Tarea que se encarga de la gestión de las cintas y del retenedor.
* @param void * parameter -> puntero a la estructura 'estado_cinta'
* @return void
*/
void gestion_cintas_task(void * parameter)
{ ... } /* gestion_cintas_task() */

/*****
/#!
* @brief Esta tarea lee los valores de los siguientes buffers:
* - 'buffer_tiempos_de_llenado_caja_s'
* - 'buffer_tiempos_de_llenado_caja_m'
* - 'buffer_tiempos_de_llenado_caja_l'
*
* Cuando alguno de ellos está lleno, calcula el promedio de
* todos los valores del buffer y publica el resultado vía MQTT.
* @param void * parameter -> &estado_cinta
* @return void
*/
void promedio_tiempos_llenado(void * parameter)
{ ... } /* promedio_tiempos_llenado() */

```

h_comunicaciones.ino

Las cabeceras y las explicaciones de las funciones de este fichero no difieren con el [h_comunicaciones.ino de la ESP32-01](#), pero la implementación cambia, ya que son otros los *topics* a los que se debe suscribir esta segunda ESP32, al igual que la gestión de los mensajes que lleguen también cambian. Para más información sobre el código se sugiere que se visite el [Repositorio de la ESP32-02 \(GitHub\)](#).

s_setup.ino

```

/**
* @file s_setup.ino
* @brief Definición e implementación de la función de inicialización.
*/

/*****
/#!
* @brief En esta función además se crean las tareas de (g_tareas.ino).
* - retenedor_camara_task()
* - gestion_cintas_task()

```

```

*           - promedio_tiempos_llenado()
* @param void
* @return void
*/
void on_setup(void)
{ ... } /* on_setup() */

```

w_loop.ino

```

/**
 * @file w_loop.ino
 * @brief Definición e implementación de la función de bucle.
 */

/*****
/*!
 * @brief Función de bucle vacío, no es necesario realizar alguna acción más
 *         aparte de las que realizan las tareas y la función 'mqttCallback'.
 * @param void
 * @return void
 */
void on_loop(void)
{ ... } /* on_loop() */

```

CÓMO EJECUTAR LA SOLUCIÓN

Para ejecutar la solución de la segunda ESP32, los pasos a seguir son los mismos que se especifican en el apartado sobre [Cómo Ejecutar la Solución de la ESP32-01](#). Hay que tener en cuenta que las [dependencias requeridas](#) para la ESP32-02 tal vez difieran con la ESP32-01.

ROBODK. SCRIPTS DE PYTHON

LENGUAJE DE PROGRAMACIÓN Y DEPENDENCIAS REQUERIDAS

El lenguaje de programación utilizado para el desarrollo de esta solución es Python, y se ha programado a través del editor [VSCodium](#), integrado en [RoboDK](#). Para que el código de los *scripts* funcione correctamente, es necesario tener instalado el simulador de RoboDK, aunque por defecto no incluye las librerías necesarias para realizar conexiones indirectas vía MQTT. Para ello se usan ciertas importaciones de librerías disponibles con la instalación base de RoboDK y algunas otras añadidas por el usuario.

Librerías disponibles con la instalación base de RoboDK

- [robodk.robolink](#) por *RoboDK*.
- [robodk.robomath](#) por *RoboDK*.
- [numpy](#) por *NumPy Developers* (versión 1.8.0).

Librerías añadidas

- [paho.mqtt.client](#) por *Roger Light and others* (versión 2.0.0).

Ficheros de desarrollo propio

- `functions.py` por Grupo PR2-A04.
- `esp32_mqtt_listener.py` por Grupo PR2-A04.
- `cobot_mqtt_listener.py` por Grupo PR2-A04.
- `industrial_mqtt_listener.py` por Grupo PR2-A04.
- `delete_box_s.py` por Grupo PR2-A04.
- `delete_box_m.py` por Grupo PR2-A04.
- `delete_box_l.py` por Grupo PR2-A04.

FICHEROS, IMPORTACIONES NECESARIAS Y FUNCIONES PRINCIPALES

Hay un total de siete *scripts* de Python, de los cuales tres de ellos, los “mqtt_listener”, son los que están en constante ejecución para que la solución funcione correctamente. El resto de los scripts contienen funciones que en algún momento de la ejecución serán necesarias.

Esta solución proporciona la conexión vía MQTT del mundo real con la simulación y viceversa. Por ejemplo, si se detecta presencia al inicio de la cinta, el *cobot* del simulador no puede montar ninguna caja hasta que se libere el espacio de *place*. Por otra parte, cuando el simulador notifique que una nueva caja ha sido llenada, las ESP32 podrán soltar el retenedor y activar las cintas (asimismo, dichas acciones se ven reflejadas en la simulación).

A continuación, se muestra una breve descripción de los ficheros, importaciones necesarias y de sus funciones. Para más información sobre el código se sugiere que se visite el [Repositorio de Scripts de RoboDK \(GitHub\)](#).

functions.py

```
"""
En este script se definen las funciones necesarias para el cobot (assemble_box)
y para el robot industrial (fill_box).
```

```
Subprogramas para (fill_box): (copy_object) y (pick_dispositivo).
"""
```

```
# ----- #
```

```
# IMPORTACIONES NECESARIAS
```

```
from json.__init__ import loads as json_loads
import numpy as np
```

```
from robodk.robolink import ITEM_TYPE_FRAME, ITEM_TYPE_PROGRAM
from robodk.robolink import ITEM_TYPE_TARGET, ITEM_TYPE_OBJECT
```

```
from robodk.robolink import Robolink, RUNMODE_SIMULATE
```

```
from robodk.robomath import xyzrpw_2_pose
```

```
# ----- #
```

```

# VINCULAR ROBOTS DE ROBODK -> ITEM_TYPE_ROBOT
...
# ----- #
# VINCULAR CINTAS (AGV - Cajas) DE ROBODK -> ITEM_TYPE_ROBOT
...
# ----- #
# VINCULAR HERRAMIENTAS DE ROBODK -> ITEM_TYPE_TOOL
...

```

```

def stop_all():
    """
    Esta función sirve para detener el cobot, el
    robot industrial y las cintas en la simulación.
    """
    ...
    ### end def stop_all() ###

```

```

def assemble_box(msg):
    """
    Esta función analiza el mensaje recibido (en concreto la información sobre
    el tipo de caja) y ejecuta el subprograma de Montaje de Caja + Transporte
    correspondiente.
    """
    ...
    ### end def assemble_box() ###

```

```

def copy_object(object_to_copy_name, new_object_name):
    """
    Con esta función se duplica un objeto existente en la estación
    'object_to_copy_name', puede ser una caja o un dispositivo de las
    estanterías. El objeto copiado se guarda en la misma posición que
    el original con el nombre 'new_object_name'.
    """
    ...
    ### end def copy_object() ###

```

```

def pick_dispositivo(id_dispositivo):
    """
    Esta función ejecuta el pick del dispositivo indicado 'id_dispositivo'.
    """
    ...
    #####
    ## FASE 1. APROXIMACIÓN PARA EL PICK ##
    #####
    ...
    #####
    ## FASE 2. PICK DEL DISPOSITIVO ##
    #####
    ...
    #####
    ## FASE 3. IR A LA POSICIÓN DE PRE-PLACE ##

```

```
#####
...
### end def pick_dispositivo() ###

def fill_box(msg):
    """
    Esta función analiza el mensaje recibido (tipo de caja, cantidad de
    dispositivos de la caja, los tipos de dispositivo y su posición de place)
    y ejecuta el Pick & Place de cada dispositivo dentro de la caja.

    Subprogramas para (fill_box): (copy_object) y (pick_dispositivo).
    """
    ...
    ### end def fill_box() ###
```

esp32_mqtt_listener.py

```
"""
Este script ajusta el valor de ciertas variables de RoboDK para que sean
acordes con la realidad, para ello se aprovechan los mensajes publicados
por la ESP32-01 y la ESP32-02 (sensores y actuadores).

Además cuando se envía una orden al robot industrial o al cobot, se publica
el estado del robot a "ocupado". Cuando los robots acaben las ordenes
publicaran automáticamente el estado del robot a "libre".
"""

# ----- #
# IMPORTACIONES NECESARIAS

from json.__init__ import loads as json_loads
import paho.mqtt.client as mqtt
from robodk.robolink import Robolink
from functions import stop_all

# ----- #
# PARÁMETROS Y TOPICS PARA LA CONEXIÓN CON MQTT
...

# ----- #
def on_message(mqttc, obj, msg):
    """
    Función que analiza el topic recibido y realiza la acción oportuna.
    """
    ...
    if msg.topic == SENSOR_INICIO_CINTA_TOPIC:
        # Cuando la ESP32-01 notifica que hay una caja al inicio de la cinta,
        # se ajusta la variable 'sensor_inicio_cinta' de RoboDK para que sea
        # acorde con la realidad:
        ...
    elif msg.topic == SENSOR_FINAL_CINTA_TOPIC:
```

```

# Cuando La ESP32-01 notifica que hay una caja al final de La cinta,
# se ajusta La variable 'sensor_final_cinta' de RoboDK para que sea
# acorde con La realidad:
...
elif msg.topic == ACCION_RETENEDOR_TOPIC:
# Cuando La ESP32-02 acciona o Libera el retenedor, se ajusta La variable
# 'accion_retenedor' de RoboDK para que sea acorde con La realidad:
...
elif msg.topic == ACCION_CINTA_01_TOPIC:
# Cuando La ESP32-02 pone en marcha o detiene La cinta 1, se ajusta
# La variable 'accion_cinta_01' de RoboDK para que sea acorde con
# La realidad:
...
elif msg.topic == ACCION_CINTA_02_TOPIC:
# Cuando La ESP32-02 pone en marcha o detiene La cinta 2, se ajusta
# La variable 'accion_cinta_02' de RoboDK para que sea acorde con
# La realidad:
...
elif msg.topic == ORDEN_COBOT_TOPIC:
# Cuando el cobot recibe una orden, se publica
# su nuevo estado, que pasa a ser "ocupado":
...
elif msg.topic == ORDEN_INDUSTRIAL_TOPIC:
# Cuando el robot industrial recibe una orden, se
# publica su nuevo estado, que pasa a ser "ocupado":
...
elif msg.topic == INDUSTRIAL_AVISO_TOPIC:
# Cuando el robot industrial Llena una caja y Lo notifica,
# se ajusta La variable 'caja_llena' de RoboDK:
...
elif msg.topic == PULSADOR_EMERGENCIA_TOPIC:
# Cuando el pulsador de emergencia sea presionado, se
# detienen todos Los elementos de La simulación de RoboDK, stop_all():
...
### end def on_message() ###

```

```

# ----- #
# CREAR CLIENTE Y ESTABLECER CONEXIÓN MQTT (BUCLE)
...

```

cobot_mqtt_listener.py

```

"""
Este script atiende a Las ordenes enviadas por MQTT dirigidas al cobot.
"""
# ----- #
# IMPORTACIONES NECESARIAS

```



```

import paho.mqtt.client as mqtt
from functions import assemble_box

# ----- #
# PARÁMETROS Y TOPICS PARA LA CONEXIÓN CON MQTT
...

def on_message(mqttc, obj, msg):
    """
    Cuando llega un mensaje a través del topic ORDEN_COBOT_TOPIC, se llama a la
    función assemble_box(), pasándole como parámetro msg sin parsear, ya que el
    mensaje se analizará dentro de esa función.
    """
    ...
    ### end def on_message() ###

# ----- #
# CREAR CLIENTE Y ESTABLECER CONEXIÓN MQTT (BUCLE)
...

```

industrial_mqtt_listener.py

```

"""
Este script atiende a las ordenes enviadas por MQTT dirigidas al robot industrial.
"""

# ----- #
# IMPORTACIONES NECESARIAS

import paho.mqtt.client as mqtt
from functions import fill_box

# ----- #
# PARÁMETROS Y TOPICS PARA LA CONEXIÓN CON MQTT
...

def on_message(mqttc, obj, msg):
    """
    Cuando llega un mensaje a través del topic ORDEN_INDUSTRIAL_TOPIC, se
    llama a la fill_box(), pasándole como parámetro msg sin parsear, ya que
    el mensaje se analizará dentro de esa función.
    """
    ...
    ### end def on_message() ###

# ----- #
# CREAR CLIENTE Y ESTABLECER CONEXIÓN MQTT (BUCLE)
...

```

delete_box_s.py, delete_box_m.py, delete_box_l.py ... delete_box_x.py

```

"""
Este script borra el contenido del AGV - Caja X tras haber sido llenada. Con

```

esta acción se consigue un efecto visual de desaparición de las cajas que llegan al final de la cinta 2. Además, cuando se borran todos los ítems del interior, se libera memoria del programa al borrar ítems innecesarios.

Después de borrar el contenido del AGV - Caja X, este vuelve a estar operativo y listo para el llenado de una nueva caja.

"""

```
# ----- #  
# IMPORTACIONES NECESARIAS  
...
```

```
# ----- #  
# VINCULAR OBJETOS Y BORRARLOS, si no existen, se captura la excepción.  
...
```

CÓMO EJECUTAR LA SOLUCIÓN

Para ejecutar la solución es necesario tener instalado el programa de RoboDK con una licencia de prueba gratuita o completa, además de tener acceso a las [dependencias requeridas](#) mencionadas anteriormente.

El proceso es sencillo, ya que una vez abierto el proyecto de la simulación en RoboDK, simplemente hay que seleccionar la opción de "Ejecutar script de Python" para cada uno de los *mqtt_listeners*, consultar Figura 3.2.

Comprobar que la conexión a internet es correcta.

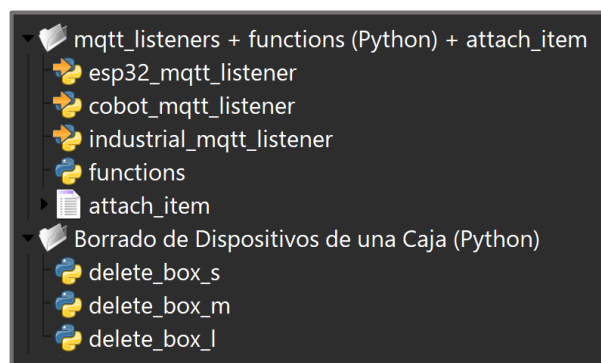


Figura 3.2: Scripts mqtt_listener en ejecución.

ALGORITMO DE COLOCACIÓN DE DISPOSITIVOS (C++)

LENGUAJE DE PROGRAMACIÓN Y DEPENDENCIAS REQUERIDAS

El lenguaje de programación utilizado para el desarrollo de esta solución es C++, a través del entorno [MVSC](#). La solución aportada es considerada una versión preliminar del producto final.

El *software* definitivo implementaría algoritmos de programación avanzada, en concreto de programación dinámica, de esta forma se conseguiría calcular una distribución de los dispositivos óptima. Debido al contexto del proyecto, que se trata de una propuesta de automatización, se ha decidido implementar una solución rápida mediante programación voraz, con la finalidad de conseguir un resultado válido para las pruebas de funcionamiento. Si la propuesta se llevara a cabo, se empezaría a trabajar en la mejora del algoritmo.

Se ha escogido el entorno de MVSC ya que este permite de forma muy cómoda la instalación de extensiones para las soluciones, como por ejemplo, tener la posibilidad de comunicar esta solución vía MQTT, o herramientas para poder insertar contenido en la base de datos creada con PostgreSQL. Estas funcionalidades no se han implementado, pero formarían parte de la solución final.

Para que el código del algoritmo funcione correctamente, es necesario incluir ciertas librerías como *iostream*, *list*, *ostream* y *string*, además del uso de algunos ficheros desarrollados por otros usuarios.

Ficheros de desarrollo propio

- **logger.h** por Grupo PR2-A04 (versión 0.6).
- **defines.h** por Grupo PR2-A04 (versión 0.6).
- **space_t.h** por Grupo PR2-A04 (versión 0.6).
- **item_t.h** por Grupo PR2-A04 (versión 0.6).
- **box_t.h** por Grupo PR2-A04 (versión 0.6).
- **main.cpp** por Grupo PR2-A04 (ejemplo de uso).

FICHEROS Y FUNCIONES PRINCIPALES

Hay un total de seis ficheros, de los cuales cinco de ellos son los principales: *logger.h*, *defines.h*, *space_t.h*, *item_t.h* y *box_t.h*. Además de proporciona fichero de ejemplo de uso (*main.cpp*), en el que se definen tres casos diferentes, uno por cada tipo de caja.

A continuación, se muestra una breve descripción de los ficheros y de sus funciones. Para más información sobre el código se sugiere que se visite el [Repositorio del Algoritmo de Colocación de Dispositivos \(GitHub\)](#).

logger.h

```
/**
 * @file      Logger.h
 *
 * @brief     Implementación y definición de Las funciones de Logger.
 *
 * En este fichero se definen hasta seis niveles diferentes para realizar
 * La depuración del código por La terminal. Para cada nivel hay disponibles
 * dos versiones, imprimir un mensaje pasando a La línea siguiente o
 * manteniéndose en La misma línea.
 *
 * @version  0.6   (2024/06/06) Traducción de Los comentarios al español
 * @version  0.5   (2024/05/07) Solucionado bug en La clase box_t "bucle infinito"
 * @version  0.4.1 (2024/05/06) Añadido cálculo de TCP pose de todos Los items
 * @version  0.4   (2024/04/28) Añadida función (generate_mqtt_order)
 * @version  0.3.2 (2024/04/27) Añadido cálculo de TCP pose de PULSERA y RELOJ
 * @version  0.3.1 (2024/04/26) Añadido cálculo de TCP pose de TELEFONO y TABLET
 * @version  0.2   (2024/03/20) Añadido funciones de Log
 * @version  0.1   (2024/03/15) Prototipo inicial funcional
 *
 * @author   Grupo PR2-A04' <mbelmar@etsinf.upv.es>
 *
 * @date     Junio, 2024
 * @section  PR2-GIIROB
 */
```

defines.h

```
/**
 * @file    defines.h
 *
 * @brief   Definiciones de Los tipos enumerados y estructuras para el proyecto.
 *
 * @version 0.6   (2024/06/06) Traducción de Los comentarios al español
 * @version ...
 *
 * @author  Grupo PR2-A04' <mbelmar@etsinf.upv.es>
 *
 * @date    Junio, 2024
 * @section PR2-GIIROB
 */

#ifndef DEFINES_T
#define DEFINES_T

typedef enum
{ ... } itemType_t;

typedef enum
{ ... } boxType_t;

typedef struct
{ ... } point_t;

#endif /* DEFINES_T */

/** end of file */
```

space_t.h

```
/**
 * @file    space_t.h
 *
 * @brief   Dados dos puntos, se define un volumen en el espacio.
 *
 * @version 0.6   (2024/06/06) Traducción de Los comentarios al español
 * @version ...
 *
 * @author  Grupo PR2-A04' <mbelmar@etsinf.upv.es>
 *
 * @date    Junio, 2024
 * @section PR2-GIIROB
 */
```

```

/*****
/!
 * @brief Primer constructor de la clase space_t.
 * @param void
 */
space_t(void)
{ ... } /* space_t() */

```

```

/*****
/!
 * @brief Segundo constructor de la clase space_t.
 * @param x0 Coordenada x del punto mínimo.
 * @param y0 Coordenada y del punto mínimo.
 * @param z0 Coordenada z del punto mínimo.
 * @param x1 Coordenada x del punto máximo.
 * @param y1 Coordenada y del punto máximo.
 * @param z1 Coordenada z del punto máximo.
 */
space_t(uint16_t x0, uint16_t y0, uint16_t z0,
        uint16_t x1, uint16_t y1, uint16_t z1)
{ ... } /* space_t() */

```

```

/*****
/!
 * @brief El destructor de clase space_t.
 * @param void
 */
~space_t(void)
{ ... } /* ~space_t() */

```

```

/*****
/!
 * @brief Establece las coordenadas del objeto space_t.
 * @param x0 Coordenada x del punto mínimo.
 * @param y0 Coordenada y del punto mínimo.
 * @param z0 Coordenada z del punto mínimo.
 * @param x1 Coordenada x del punto máximo.
 * @param y1 Coordenada y del punto máximo.
 * @param z1 Coordenada z del punto máximo.
 * @return void
 */
void set_space(uint16_t x0, uint16_t y0, uint16_t z0,
              uint16_t x1, uint16_t y1, uint16_t z1)
{ ... } /* set_space() */

```

```

/*****
/!
 * @brief Indica si el objeto que llama a la función es un
 * subconjunto del parámetro de entrada (otherSpace).
 * @param otherSpace El espacio a comparar.
 * @return Devuelve verdadero o falso.

```

```

*/
bool is_subset_of(space_t otherSpace)
{ ... } /* is_subset_of() */

/*****/
/#!/
* @brief Indica si el objeto que llama a La función está
* formado por el parámetro de entrada (otherSpace).
* @param otherSpace El espacio a comparar.
* @return Devuelve verdadero o falso.
*/
bool is_made_up_of(space_t otherSpace)
{ ... } /* is_made_up_of() */

/*****/
/#!/
* @brief Agrega un desplazamiento a un espacio.
* @param space El espacio a desplazar.
* @param displacement La cantidad que se quiere desplazar.
* @return El nuevo espacio desplazado.
*/
friend space_t operator+(const space_t & space, const point_t & displacement)
{ ... } /* operator+() */

/*****/
/#!/
* @brief La función para saber si space1 es lo mismo que space2.
* @param space1 El primer espacio a comparar.
* @param space2 El segundo espacio a comparar.
* @return Devuelve verdadero o falso.
*/
friend bool operator!=(const space_t & space1, const space_t & space2)
{ ... } /* operator!=() */

/*****/
/#!/
* @brief Convierte las coordenadas space_t en una string.
* @param void
* @return String con los valores de las coordenadas.
*/
string space_to_str(void)
{ ... } /* space_to_str() */

/*****/
/#!/
* @brief Devuelve el valor del atributo min.x.
* @param void
* @return Coordenada x del punto mínimo.
*/
uint16_t min_x(void)
{ ... } /* min_x() */

```

```

/*****
/#!
 * @brief Devuelve el valor del atributo min.y.
 * @param void
 * @return Coordenada y del punto mínimo.
 */
uint16_t min_y(void)
{ ... } /* min_y() */

```

```

/*****
/#!
 * @brief Devuelve el valor del atributo min.z.
 * @param void
 * @return Coordenada z del punto mínimo.
 */
uint16_t min_z(void)
{ ... } /* min_z() */

```

```

/*****
/#!
 * @brief Devuelve el valor del atributo max.x.
 * @param void
 * @return Coordenada x del punto máximo.
 */
uint16_t max_x(void)
{ ... } /* max_x() */

```

```

/*****
/#!
 * @brief Devuelve el valor del atributo max.y.
 * @param void
 * @return Coordenada y del punto máximo.
 */
uint16_t max_y(void)
{ ... } /* max_y() */

```

```

/*****
/#!
 * @brief Devuelve el valor del atributo max.z.
 * @param void
 * @return Coordenada z del punto máximo.
 */
uint16_t max_z(void)
{ ... } /* max_z() */

```

item_t.h

```

/**
 * @file    item_t.h
 *
 * @brief   Implementación y definición de la clase item_t.

```

```

*
* @version 0.6 (2024/06/06) Traducción de Los comentarios al español
* @version ...
*
* @author Grupo PR2-A04' <mbelmar@etsinf.upv.es>
*
* @date Junio, 2024
* @section PR2-GIIROB
*/

/*****/
/#!
* @brief El constructor de la clase item_t.
* @param item_id Indica qué tipo/modelo/variante de item es.
*/
item_t(string item_id)
{ ... } /* item_t() */

/*****/
/#!
* @brief El destructor de la clase item_t.
* @param void
*/
~item_t(void)
{ ... } /* ~item_t() */

/*****/
/#!
* @brief Devuelve el atributo de size item_t.
* @param void
* @return size EL atributo privado que indica el tamaño del item.
*/
space_t get_size(void)
{ ... } /* get_size() */

/*****/
/#!
* @brief Devuelve el atributo posInBox de item_t.
* @param void
* @return EL valor de la posición del item en la caja.
*/
space_t get_posInBox(void)
{ ... } /* get_posInBox() */

/*****/
/#!
* @brief Método para modificar el atributo privado posInBox de item_t.
* @param posInBox La variable space_t que se guardará en posInBox.
* @return void
*/
void set_posInBox(space_t newPosInBox)

```



```

{ ... } /* set_posInBox() */

/*****
/!
* @brief Método para modificar el atributo privado target_str de item_t.
* @param target_str La pose de TCP calculada.
* @return void
*/
void set_target_str(string target_str)
{ ... } /* set_target_str() */

/*****
/!
* @brief Devuelve el atributo type de item_t.
* @param void
* @return EL tipo de item.
*/
itemType_t get_type(void)
{ ... } /* get_type() */

/*****
/!
* @brief Devuelve el atributo target_str de item_t.
* @param void
* @return EL objetivo TCP.
*/
string get_target_str(void)
{ ... } /* get_target_str() */

/*****
/!
* @brief Devuelve el atributo item_id de item_t.
* @param void
* @return EL identificador del item.
*/
string get_item_id(void)
{ ... } /* get_item_id() */

```

box_t.h

```

/**
* @file    box_t.h
*
* @brief   Implementación y definición de La clase box_t.
*
* @version 0.6 (2024/06/06) Traducción de Los comentarios al español
* @version ...
*
* @author  Gurpo PR2-A04' <mbelmar@etsinf.upv.es>
*
* @date    Junio, 2024

```

```

* @section PR2-GIIOB
*/

/*****/
/#!/
* @brief EL constructor de la clase box_t.
* @param type Indica qué tipo de caja es.
* @param itemsToPlaceInOrder Lista de elementos a colocar en la caja.
*/
box_t(boxType_t type, list<item_t> * itemsToPlaceInOrder)
{ ... } /* box_t() */

/*****/
/#!/
* @brief EL destructor de la clase box_t.
* @param void
*/
~box_t(void)
{ ... } /* ~box_t() */

/*****/
/#!/
* @brief Determina si newPlaceSpace es una posición válida, es decir, no
* se sale de la caja y no se encuentra en volumen ya ocupado.
* @param newPlaceSpace EL nuevo espacio a verificar.
* @return Verdadero o falso.
*/
bool is_valid_space(space_t newSpace)
{ ... } /* is_valid_space() */

/*****/
/#!/
* @brief Encuentra un nuevo punto donde se colocará el siguiente elemento.
* @param void
* @return Nuevo punto que indica el siguiente lugar de origen.
*/
point_t search_newOriginPoint(void)
{ ... } /* search_newOriginPoint() */

/*****/
/#!/
* @brief Busca posibles uniones de volumen que puedan existir entre el aux
* e it, y viceversa. Se realizan operaciones para sumar el volumen
* de espacio ocupado.
* @param *it Puntero a un iterador de la lista spaceInUse.
* @param *aux El puntero a space_t que se desea
* agregar a la lista spaceInUse.
* @param *spaceIsAdded Puntero a la variable bool.
* @return void
*/
void search_possible_unions(space_t * aux, list<space_t>::iterator * it,

```

```

        bool * needsToBeAdded)
{ ... } /* search_possible_unions() */

/*****
/*!
 * @brief Actualiza el espacio actual en uso de la caja.
 * @param toAdd Nuevo espacio en uso para ser añadido.
 * @return void
 */
void update_spaceInUse(space_t toAdd)
{ ... } /* update_spaceInUse() */

/*****
/*!
 * @brief Coloca todos los elementos de itemsToPlace dentro de la caja y
 * actualiza la lista placedItems. También establece targetPlace
 * de todos los elementos.
 * @param void
 * @return void
 */
void place_items_in_box(void)
{ ... } /* place_items_in_box() */

/*****
/*!
 * @brief Este método calcula las poses de TCP para todos los elementos
 * para que puedan colocarse correctamente en el simulador RoboDK.
 * @param void
 * @return void
 */
void calculate_TCP_poses(void)
{ ... } /* calculate_TCP_poses() */

/*****
/*!
 * @brief Este método genera una orden en formato JSON para enviársela
 * al robot industrial del simulador RoboDK (vía MQTT).
 * @param void
 * @return void
 */
void generate_mqtt_order(void)
{ ... } /* generate_mqtt_order() */

/*****
/*!
 * @brief Devuelve el atributo mqtt_order de box_t.
 * @param void
 * @return La orden en formato JSON.
 */
string get_mqtt_order(void)
{ ... } /* get_mqtt_order() */

```

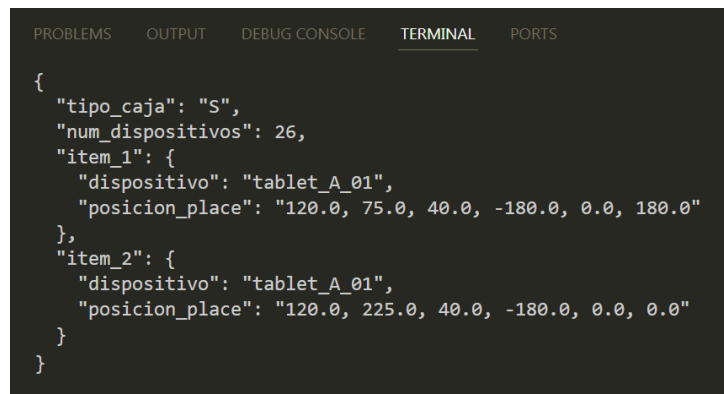
main.cpp

```
/**
 * @file    main.cpp
 *
 * @brief   Tres ejemplos de uso, uno por cada tipo de caja. Dada una Lista con
 *          dispositivos, se calculan sus posiciones de place dentro de caja y
 *          se genera la orden (formato JSON) para el robot industrial.
 *
 * @version 0.6   (2024/06/06) Traducción de los comentarios al español
 * @version ...
 *
 * @author  Grupo PR2-A04' <mbelmar@etsinf.upv.es>
 *
 * @date    Junio, 2024
 * @section PR2-GIIROB
 */
```

CÓMO EJECUTAR LA SOLUCIÓN

Para ejecutar la solución hay que generar un fichero ejecutable del archivo *main.cpp*, por lo que es necesario tener un compilador de C++ instalado.

En el caso de estar trabajando en el entorno de MVSC, simplemente configurando el proceso de compilación, se puede generar el ejecutable desde el propio entorno, consultar Figura 3.3.



```
{
  "tipo_caja": "S",
  "num_dispositivos": 26,
  "item_1": {
    "dispositivo": "tablet_A_01",
    "posicion_place": "120.0, 75.0, 40.0, -180.0, 0.0, 180.0"
  },
  "item_2": {
    "dispositivo": "tablet_A_01",
    "posicion_place": "120.0, 225.0, 40.0, -180.0, 0.0, 0.0"
  }
}
```

Figura 3.3: Ejemplo de salida por terminal del ejecutable *main*.

4. CONCLUSIONES Y TRABAJO FUTURO

A lo largo del documento se ha incidido en que las soluciones aportadas son una versión preliminar y posiblemente contengan errores. En el caso de que se llevara a cabo la propuesta de automatización, se solucionarían posibles errores y se aplicarían mejoras en el *software*.

Las posibles mejoras del *software* consisten en aprovechar al máximo el potencial de las ESP32, o en el caso del Algoritmo de Colocación de Dispositivos, pasar de obtener una solución voraz a obtenerla mediante programación dinámica. Asimismo, hay ciertos casos que no se han tenido en consideración, y que el algoritmo final debería contemplar; por ejemplo, si se realiza un pedido de 24 tablets, lo ideal sería dividir las tablets en varias cajas. En una caja tipo L podrían colocarse las 24 cajas, pero el peso total de la caja se elevaría por encima de los 25 kg, por lo que la caja podía romperse.

5. BIBLIOGRAFÍA

APLICACIONES EMPLEADAS

- [Arduino IDE 2.0](#)
- [Visual Studio Code - Microsoft](#)
- [RoboDK - Simulator for industrial robots and offline programming](#)
- [VSCodium - freely-licensed binary distribution of Microsoft's editor VS Code](#)

DOCUMENTOS DE REFERENCIA

- [Preparación de Pedidos - Proyecto de Automatización - Grupo A4](#)
- [2. RoboDK API \(robodk package\) — RoboDK API Documentation](#)
- [6. Examples — RoboDK API Documentation](#)

