Memoria final del proyecto Rito

Proyecto de Robótica

Grado de Tecnologías Interactivas

Hernández López, Fabio
Jiménez García, David
Oller Caballé, Marc
Pujol Martorell, Sergio
Marí Giner, Ignasi

Índice

Introducción

En plena pandemia, se vio un incremento de gente en los supermercados. Este hecho, complica las tareas como ir a comprar o esperar en las abarrotadas colas de los cajeros. Nuestra idea parte de este concepto, un robot que nos asista en nuestras tareas cotidianas de forma que estas se realicen de forma más rápida y automatizada, evitando los cúmulos de gente.

Tras varias propuestas como un perro guía o un robot que ayuda a personas de 3ª edad en casa, varios hemos coincidido en que la creación de un robot como carrito de compra era la opción más polivalente en cuanto a servicios y originalidad. En resumen, se trata de un robot que se mueve de forma autónoma por el supermercado en función de nuestras necesidades y escanea los productos a través del reconocimiento de imágenes.

Objetivos

A la hora de realizar el concepto de nuestro proyecto, dejamos claros los objetivos principales y que debían estar a la hora de concebir el robot.

Navegación autónoma del robot

El primero de todo, la navegación del robot. Esta implementación consiste en una navegación autónoma del robot, que permite al cliente despreocuparse del carrito y que este se ocupe de desplazarse de forma autónoma.

Interfaz web para controlarlo

Otro objetivo principal del proyecto, es la App Web para poder controlar el robot. Esta aplicación web consiste en una interfaz sencilla donde poder visualizar el mapa del supermercado, añadir listas de la compra y modificarlas, un apartado donde vincular tu teléfono al carrito del supermercado, y visualizar tu propio perfil.

Uso de OpenCV

OpenCV es una herramienta muy potente que hemos utilizado para reconocer los productos del supermercado, aunque se ha abarcado solo en el sprint 3, se ha desarrollado con un diseño simplista pero aun así, con resultados bastante eficientes, las redes convolucionales han sustituido el uso general de OpenCV, ya que los resultados mejoran exponencialmente.

Reconocimiento

El reconocimiento de objetos es un objetivo muy claro dentro de este proyecto. El robot, va a tener la capacidad de obtener imágenes de los productos de la compra, mediante su cámara, para que se pueda diferenciar entre ellos y poder automatizar aún más la compra.

Funcionalidades

1. **Reconocimiento de objetos**
   Esta funcionalidad nos permite automatizar el robot para que pueda reconocer objetos de diferentes formas, mediante OpenCV y tensor flow.

2. **Crear listas, modificar y eliminar**
   Dar la posibilidad al usuario de crear listas de la compra para poder guardarlas en la propia app web. Y a la vez, poder editarlas o eliminarlas.

3. **Iniciar sesión con usuario en la App**
   Autenticación dentro de la aplicación por usuario para disponer de los datos guardados.

4. **Visualizar mapa del supermercado**
   Disponer del mapa del supermercado en la App Web con sus zonas de compras y localización del robot para saber donde se encuentra.

5. **Vincular robot a la App Web**
   Posibilidad de vincular o desvincular el robot desde la propia app

6. **Crear ruta para recoger alimentos por el supermercado para el robot**
   Posibilidad de a través de las listas de la compra, realizar una ruta para recoger los alimentos recorriendo el supermercado.

7. **Calcular peso**
   En la interfaz del mapa de la app, el usuario tendrá la opción de visualizar el peso en el interior del carrito.

8. **Detectar un nuevo producto en el carrito**
   Mediante el peso, vamos a tener la posibilidad de visualizar que se ha añadido un nuevo producto al carrito.

9. **Mostrar la ruta en el mapa**
   Mostrar la ruta que va a llevar a cabo el robot en la interfaz del mapa.

10. **Enviar carrito a por pedido**
    Mediante las rutas, se puede enviar al robot carrito a recoger un pedido a una zona específica del supermercado, como una pescadería, carnicería, etc.

Desarrollo

Mockup del diseño web

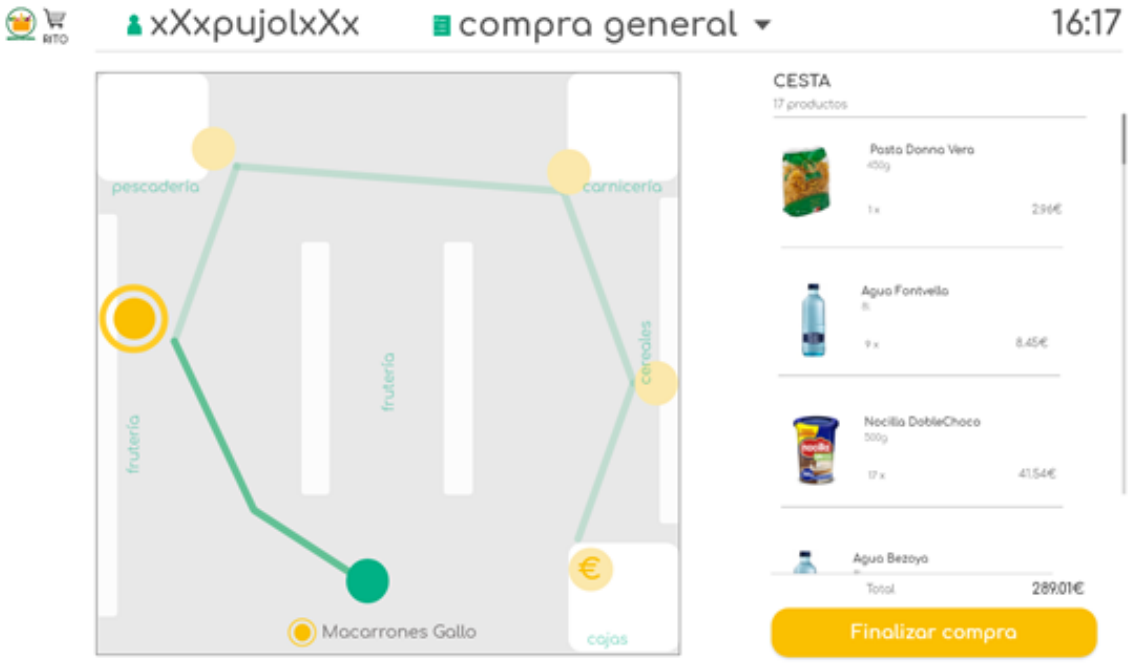
La idea inicial era implementar dos diseños diferentes de la aplicación. Un diseño para los teléfonos móviles de los clientes del supermercado en una aplicación nativa creada con tecnologías web y otro para las pantallas que irían adheridos a los carros de compra.

Diseño de la pantalla del carrito

1. Pantalla de vinculación.

En esta vista se ofrece el pin del carrito para que se conecte el usuario mediante la web en su teléfono móvil. Existe la opción de entrar como invitado para los clientes que no sean usuarios de la web o no dispongan de cuenta. Además, hay enlaces a las tiendas en las cuales se podría descargar la aplicación.

2. Pantalla de compra.



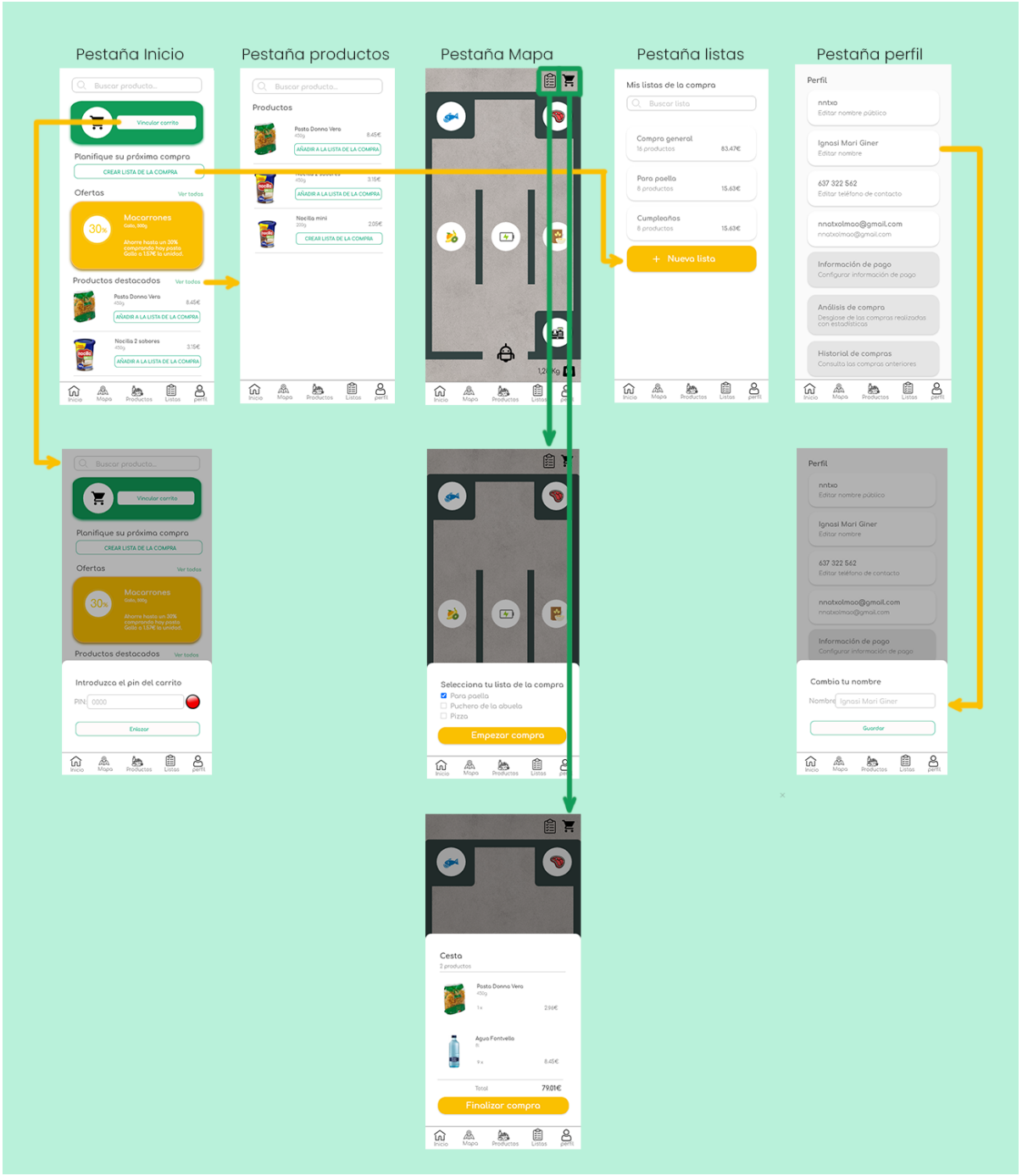
En esta vista se puede apreciar en la parte superior el nombre de usuario y un desplegable con las lista de la compra que tiene el usuario guardadas en el que se puede seleccionar qué lista de la compra quiere el usuario comprar. También se ve el mapa del supermercado en el cual se ve la posición del carrito en tiempo real y los productos y ruta que va a seguir el carrito para realizar la compra lo más rápidamente posible. A la izquierda del mapa podemos observar la Cesta en la cual se podrá ver en tiempo real cómo se van actualizando los productos que están metidos en la cesta del carrito de compra, así como el precio total y el número de productos. El botón de Finalizar compra mandaría el carro a la caja para hacer el pago.

Implementación de las pantallas en los robots

El diseño para las pantallas del carro no se ha llegado a implementar por varias razones. La primera y más importante es porque poner más peso en el robot iba a suponer problemas para la navegación del robot. Teniendo en cuenta esto y que esta pantalla tampoco ofrecía información nueva, sino que toda esa información la podíamos encontrar también desde el móvil, optamos por prescindir de ella para evitar problemas y centrarnos en las funcionalidades.

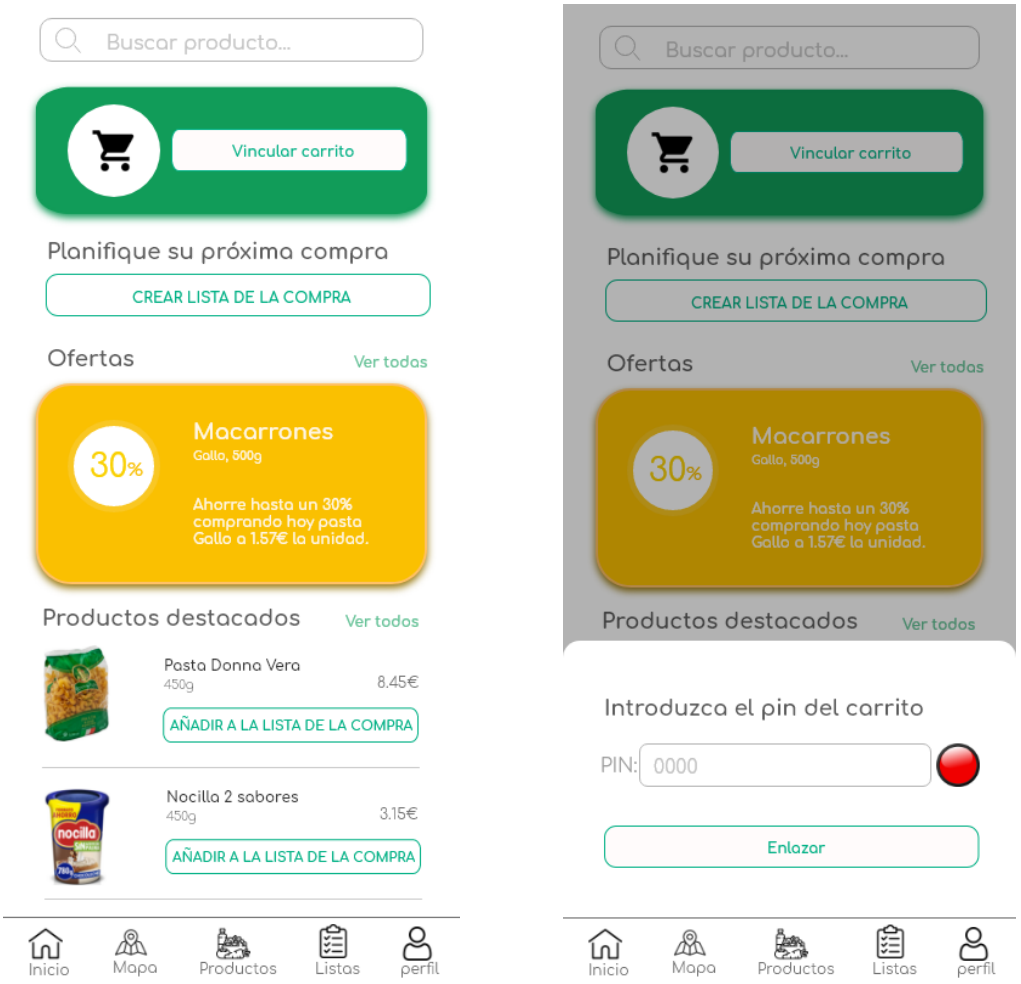
Diseño de la aplicación del cliente

A continuación de presenta el diagrama de navegación de nuestra aplicación:



Para explicar las diferentes pantallas de la aplicación vamos a hacernos servir del menú de la aplicación:

1. Pestaña Inicio



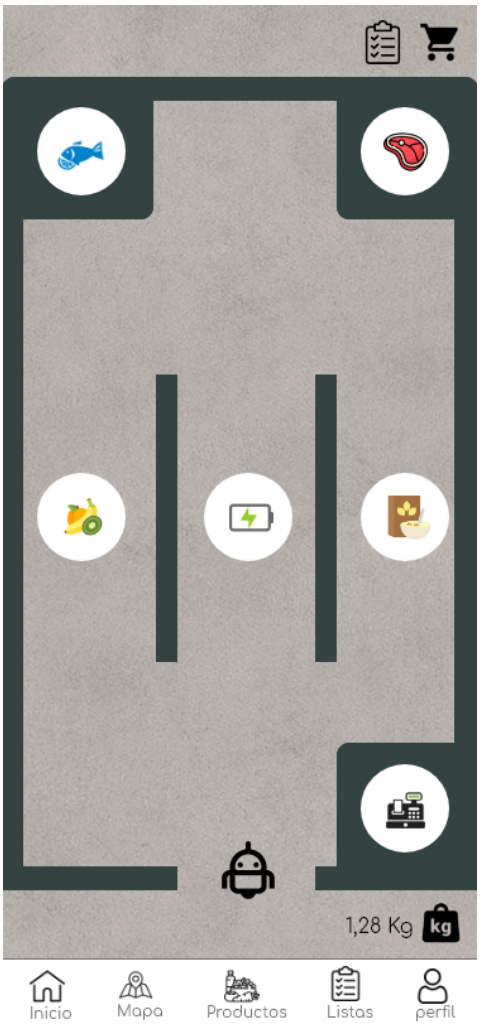
La página de inicio cuenta con unas cuantas funcionalidades. La primera es la búsqueda de productos mediante un campo de texto. Debajo tenemos un botón para que los usuarios se vinculen con el carrito que van a usar en su compra. Al pulsar el botón aparece el desplegable de la imagen anterior. Justo debajo de este botón hay otro para que los usuarios creen una nueva lista de la compra. Si el usuario pulsa este botón será redirigido a la pestaña de listas, donde podrá crear una nueva lista.

Debajo del botón tenemos una zona donde se podrán ver las ofertas que haya en ese momento y si pulsamos sobre el texto *Ver todas*, el usuario será redirigido a una vista en la cual podrá ver todos los productos. Seguidamente tenemos una sección de productos destacados los cuales podremos agregar a nuestras listas de la compra o podremos ver todos los productos pulsando sobre *Ver todos* que llevará al usuario a la pestaña de productos.

2. Pestaña Mapa

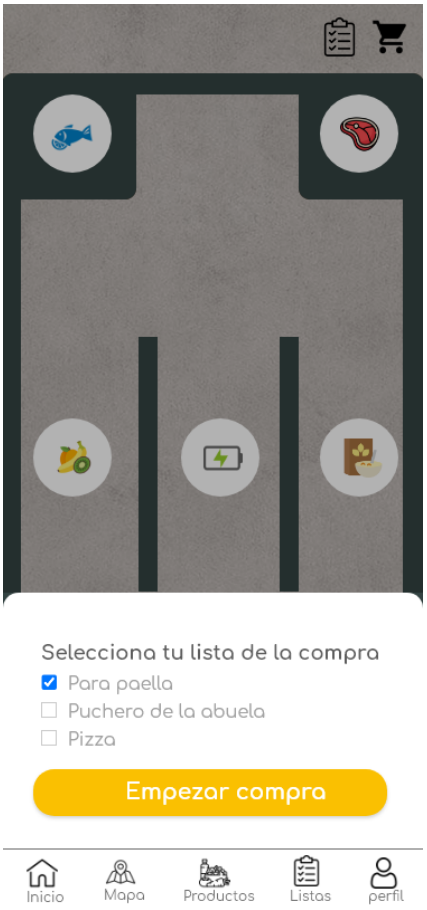
Esta pestaña es la más compleja ya que es la que el usuario utiliza cuando está en el supermercado dispuesto a realizar la compra.

2.1 Navegación y compra



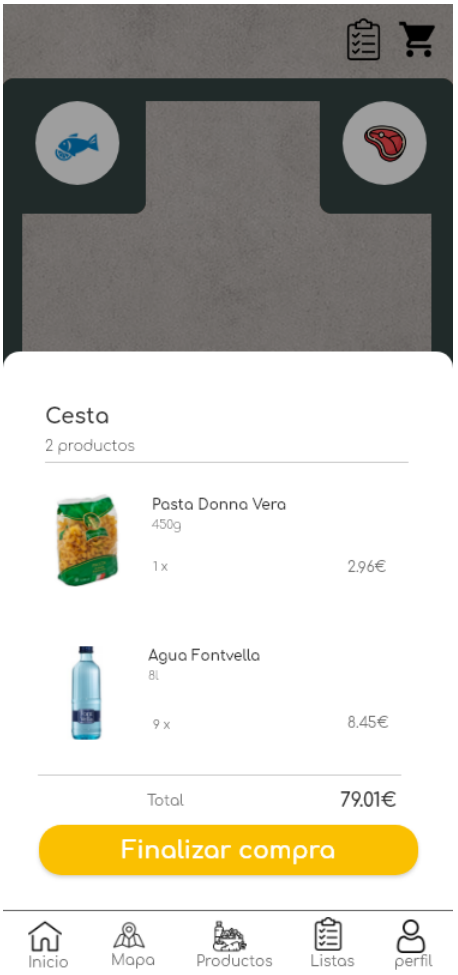
En esta pantalla el usuario puede ver el mapa del supermercado en el cual se ve la posición del carrito en tiempo real. Falta por implementar la localización de los productos en el mapa, el nombre del siguiente producto a recoger y la visualización de la ruta que va a seguir el carrito para realizar la compra lo más rápidamente posible. Arriba a la derecha tenemos el logo de las listas que accionará el desplegable de Selección de listas (2.2) donde podremos añadir o quitar las listas de la compra seleccionadas. A su derecha está el logo de la cesta. Si el usuario lo pulsa, se abrirá el desplegable de Visualización de cesta (2.3).

2.2 Selección de listas de la compra



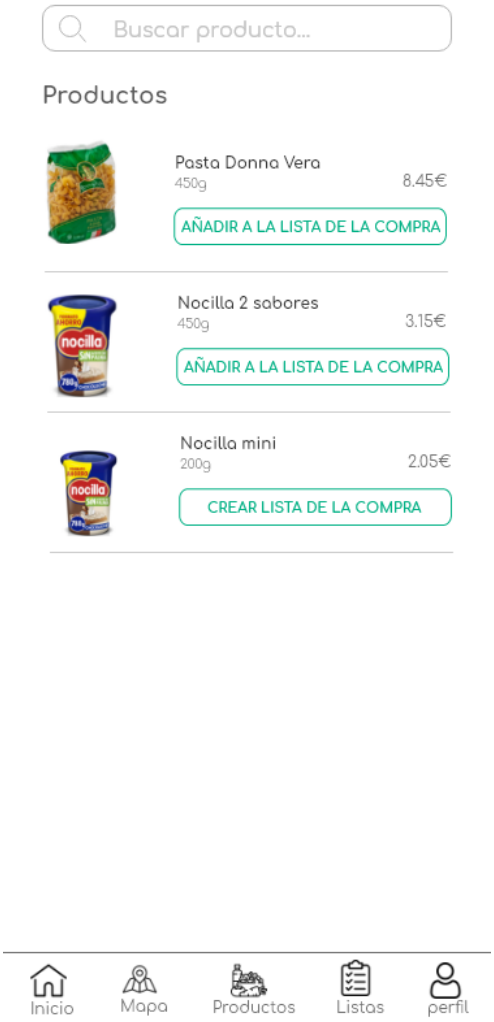
En esta pantalla el usuario puede seleccionar la lista de productos que quiere comprar. Puede seleccionar una o varias.

2.3 Visualización de la cesta



En esta pantalla el usuario podrá visualizar en tiempo real tanto los productos que han sido añadidos en el carrito como la cantidad de cada producto, el total de productos y el precio total de lo que hay en la cesta.

3. Pestaña Productos

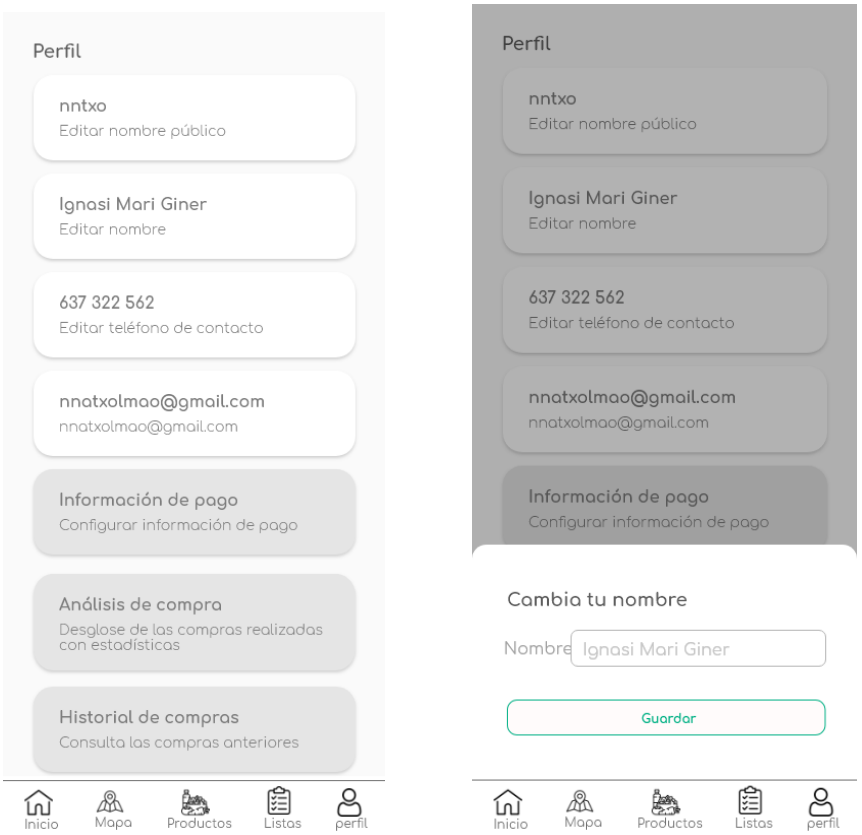


En esta pestaña podremos ver el catálogo que ofrece el supermercado. También podemos hacer búsquedas de los productos por su nombre (como en la pantalla de inicio) mediante una entrada de texto.

4. Pestaña Listas

En esta pestaña se pueden visualizar todas las listas que ha creado el usuario, identificadas por el nombre que escribió el usuario al crearlas. También se puede ver el número de productos que incluye cada lista y el precio total de la lista. En la parte superior tenemos una entrada de texto que sirve como buscador de listas por su nombre. Y en la parte inferior se encuentra un botón + Nueva lista que sirve para crear una lista nueva.

5. Pestaña Perfil



En la pestaña de perfil se puede visualizar la información personal del usuario (nombre de usuario, nombre completo, número de teléfono, correo electrónico). Esta información personal se puede editar pulsando sobre estos campos. Las funcionalidades de los dos últimos botones y de procesar pagos parecían muy interesantes de implementar, pero al final se decidió no implementarlas por falta de tiempo y recursos.

Tecnologías usadas en la web

En este apartado vamos a tratar las diferentes tecnologías utilizadas en la aplicación del cliente.

La idea inicial era dotar al usuario de una aplicación nativa para su teléfono. Para el Sprint 2 llegamos a desarrollar una aplicación que podía ser compilada para dispositivos móviles tanto Android como iOS. Para conseguir esto hicimos uso de React Native. La implementación de los diseños fue relativamente sencilla pero no se podían conectar los móviles a la red que se nos proporcionaba. Así pues, se tuvo que adaptar el código para utilizar un visualizador web (con Expo) para poder utilizar la aplicación desde un ordenador.

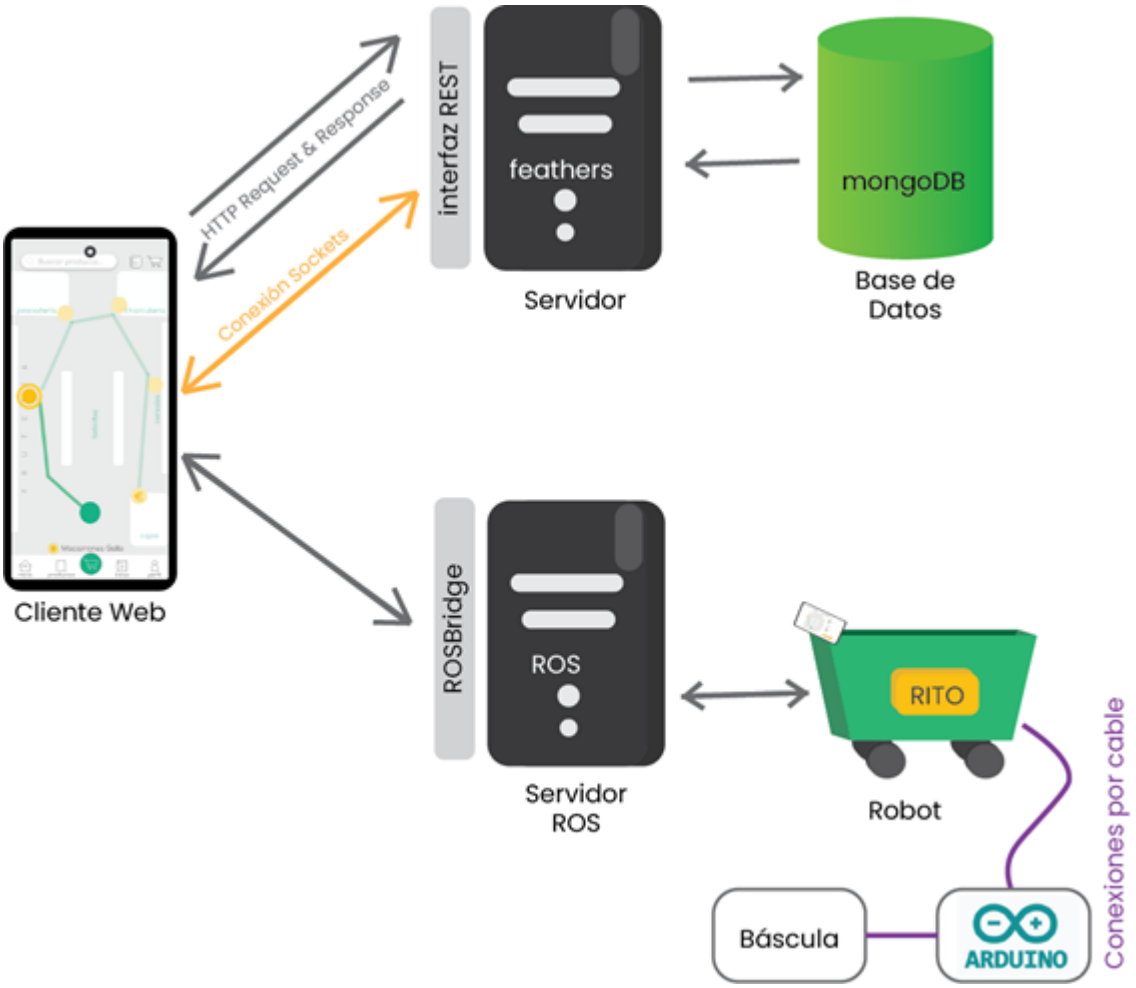
La aplicación en ordenador hacía un render de las vistas que no se correspondía con el que se había construido y hubo más problemas de conexión con Expo. Por estas razones decidimos abandonar la idea de utilizar React Native para desarrollar la aplicación.

A partir del Sprint 3 desarrollamos una aplicación web sin ningún framework de desarrollo, solamente HTML5, CSS y JavaScript.

Las librerías que se han usado han sido:

- Bootstrap 4.0: Para los modals, los desplegables y los estilos de la página web en general.
- jQuery 3.2.1: Se utiliza juntamente con Bootstrap y ha agilizado la programación en JS.
- RosLib: Para conectar con ROS.
- Socket.IO 1.7.3: Es una librería de JavaScript que permite comunicación entre cliente y servidor de forma bidireccional, en tiempo real y mediante eventos. Se ha utilizado para las conexiones en tiempo real con el servidor.
- feathers.js (Client) 2.0.0: Librería para comunicarse con el servidor Feathers. Contiene los plugins necesarios para trabajar con conexiones real-time con socket.io, con los eventos que emite el servidor y con la autentificación mediante JSON Web Tokens (JWT).
- JWT: Esta tecnología nos ha servido para la autentificación de los usuarios cuando inician sesión para mantener esta autentificación en toda la página web. Este token es suministrado por el servidor y es almacenado localmente por el cliente.
- Cookies: Se han utilizado cookies para guardar la conexión del cliente con el robot en toda la web, independientemente de la pestaña en la que estuviera el usuario. En esta cookie se guarda la IP del servidor de ROS en hexadecimal.

Esquema de los componentes



OpenCV

OpenCV, es una herramienta bastante potente que nos permite extraer información interesante de una imagen con la finalidad de procesarla para conseguir un resultado específico.

En nuestro proyecto se utilizó el reconocimiento de imágenes con OpenCV para identificar ciertos productos durante el sprint 3, ya que en el producto final, nuestro robot sería capaz de reconocer los objetos mediante una serie de redes neuronales, con un entrenamiento específico para la temática de nuestro proyecto.

Así pues, el reconocimiento con OpenCV se ha centrado en identificar diferentes colores, referenciando a un tipo de alimento en específico, por ejemplo, cuando se trataba de carne, se procesaba un objeto de color rojo, asimismo, si el pedido era de pescado, al momento que se captaba un objeto de color azul por el obturador de la cámara, el motor de reconocimiento trataba la imagen, mediante los conceptos vistos en clase.

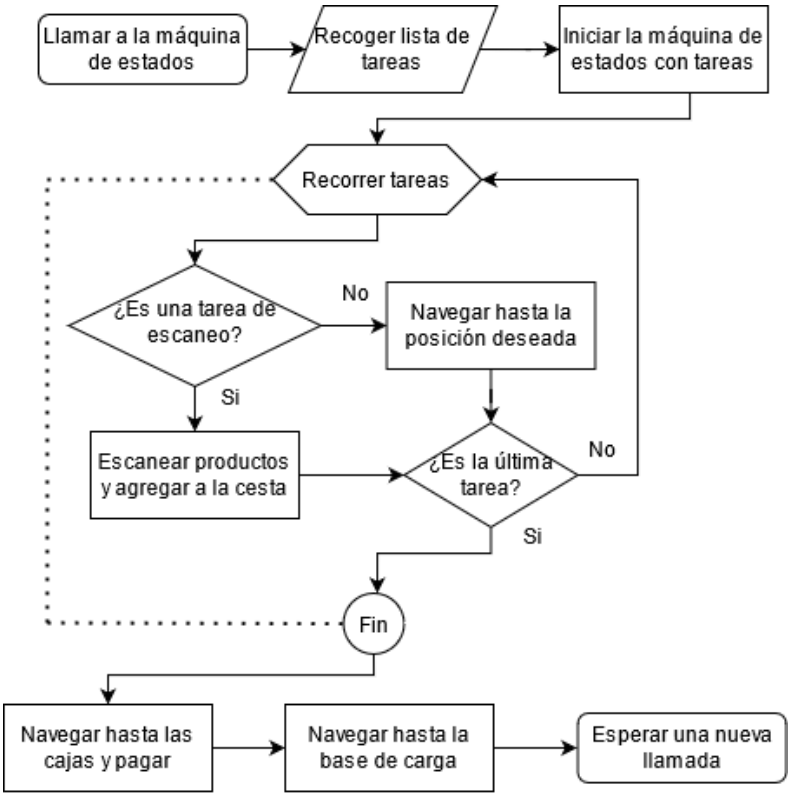
Acto seguido, después de verificar que se ha procesado un target específico correctamente, se da paso a la funcionalidad que precede.

Máquina de estados

Las máquinas de estado son un punto bastante importante de nuestro robot, y es que sin el planteamiento correspondiente de una máquina de estados eficiente, el desarrollo del autómata se concluiría con un funcionamiento indeseado.

Nuestra idea era clara y sencilla, guiar al cliente a un punto específico donde se encuentre el producto, escanearlo y llevarlo al cajero para proceder al pago. Con este planteamiento, hemos diseñado una máquina de estados escalable con la que poder agilizar el movimiento de los clientes y que tengan una experiencia lo más eficiente posible.

El procedimiento sigue el siguiente diagrama de flujo.



Como podemos observar, el diagrama tiene como nodo principal, la llamada desde el nodo maestro o desde el cliente, en este caso, el dispositivo/tablet incluida en el carrito, y publicará en el topic /activate_state la lista de las tareas que tiene que realizar.

El topic se ejecuta mediante un servicio externo a la máquina de estado, desarrollado por el grupo de trabajo y ejecutado desde el mismo archivo de lanzamiento, este servicio, se encarga de recibir la lista en tipo cadena de texto, basada en el formato csv, y devuelve el resultado con un booleano de finalización sin problemas y el coste de los productos.

```
# request
string tasks
---
# response
bool success
string bill
```

Siguiendo el recorrido del diagrama del flujo, una vez hemos obtenido todas las tareas de la lista, se procederá a recorrer el conjunto de tareas, a cada una de ellas, se revisará si es la última tarea, ya que esta, después de escanear el producto requerido, terminará la iteración y finalizará el recorrido del usuario, en la zona de cajas. Finalmente, el robot terminará su misión dirigiéndose a la zona de carga y así poder estar listo para el próximo uso.

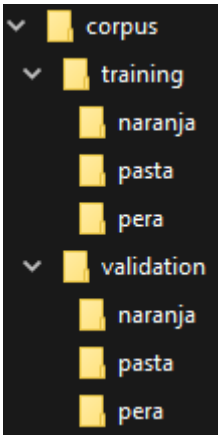
Reconocimiento

- **Definiendo el dataset**

Una vez visto e implementado la librería OpenCV en el Sprint 3, para este último Sprint iremos un paso más allá y entrenaremos una red convolucional con fotos de nuestros productos para que la cámara del robot sepa reconocerlos y los clasifique a partir del modelo entrenado.

Para ello lo primero es sacar fotos de los diferentes productos a clasificar. Lo ideal es sacar mínimo 100 fotos por cada producto en diferentes condiciones (luz, ángulos,etc).

Cuando tengamos todas las fotos las organizamos en subcarpetas (1 por cada producto). En nuestro caso hemos utilizado peras, naranjas y espagueti:



Una vez tenemos las fotos en sus respectivos directorios los metemos todos en otro llamado training, que es el que usará el programa para entrenar el modelo.

Acto seguido, fuera del directorio training, creamos el directorio validation que servirá para validar el modelo entrenado. Este tendrá las mismas subcarpetas de clases que el de training pero dentro de ellas copiaremos solo un 20-25% de las imágenes de cada subclase del

directorio training. Es decir, si tenemos 100 fotos por clase en training, copiaremos entre 20 y 25 fotos por clase en cada subdirectorio de validation para que las compruebe.

Para finalizar, metemos los directorios training y validation en uno mayor llamado corpus. Este paso solo sería necesario si el modelo se entrena en un colab, ya que habría que comprimirlo para subirlo a drive.

- **Entrenamiento con Tensor Flow**

Una vez generado el dataset usaremos las librerías Tensor Flow y Keras para entrenar, validar y testear un modelo. Para ello partiremos de la base del colab en poliformat, el cual contiene los métodos necesarios para entrenarlo y validarlo. Así que subimos el zip del corpus y lo descomprimimos en el colab en training y validation.

Acto seguido llamamos al método de training para entrenar el modelo:

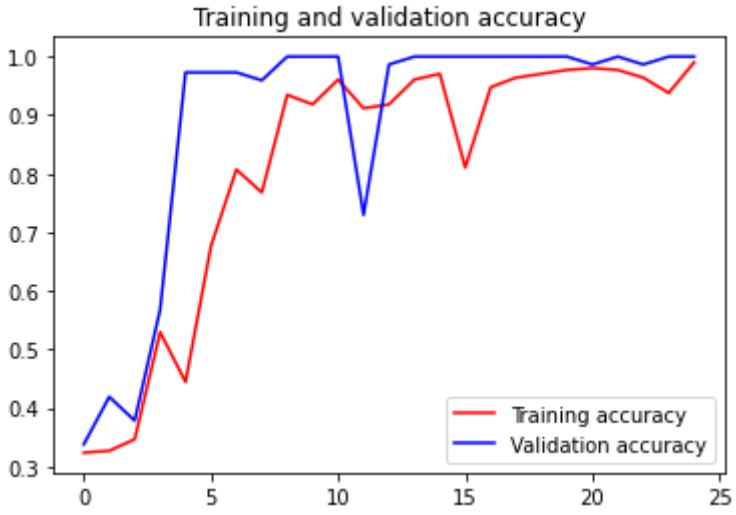
```
BATCH_SIZE = 100
WIDTH      = 250
HEIGHT     = 250

training_datagen = ImageDataGenerator(
      rescale = 1./255,
      rotation_range=5,
      width_shift_range=0.05,
      height_shift_range=0.05,
      zoom_range=0.5,
      horizontal_flip=True,
      shear_range=5)

train_generator = training_datagen.flow_from_directory(
    "/content/corpus/training/",
    batch_size=BATCH_SIZE,
    #color_mode="grayscale",
    target_size=(WIDTH, HEIGHT))
```

A continuación definimos la red convolucional y modificamos los parámetros del modelo en función de nuestras necesidades. Nosotros definimos solo 25 epochs ya que los productos son bastante diferentes entre sí (los epoch es el nº de ciclos donde los datos de entrenamiento pasan por la red neuronal para que esta aprenda sobre ellos). Lo normal sería definir 100 epochs para obtener resultados óptimos.

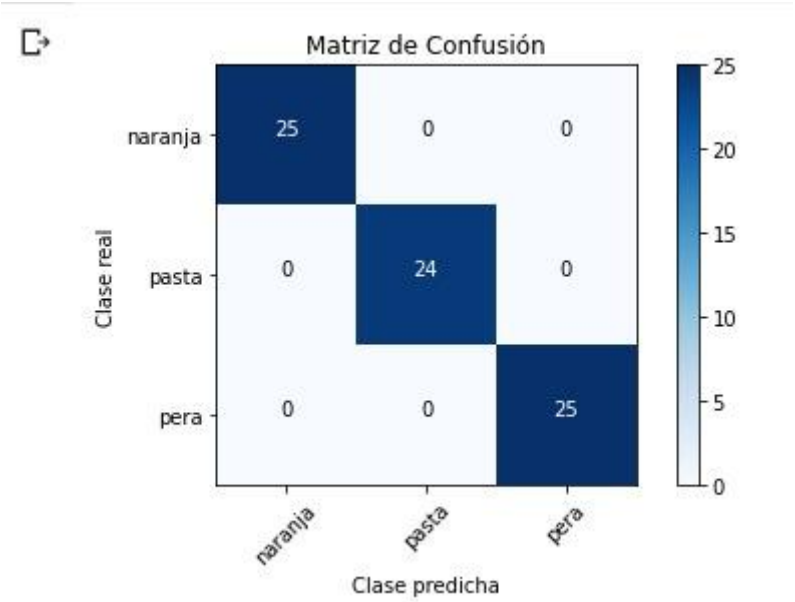
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(WIDTH, HEIGHT, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

```
history = model.fit(
    train_generator,
    batch_size = BATCH_SIZE,
    epochs=25, #100
    validation_data = validation_generator,
    validation_batch_size = BATCH_SIZE)
```

Una vez ejecutado obtenemos la gráfica con los resultados. En nuestro caso el error es mínimo:



Podemos comprobar si se mezclan algunas imágenes con la matriz de confusión, la cual permite visualizar cuantos ejemplos son correctamente clasificados y en caso de que no lo estén, con que clase se confunden:



Como podemos comprobar todas las fotos están correctamente clasificadas.

Veamos ahora si es capaz de validar correctamente las muestras ubicadas en el directorio “validation”:

```
validation_generator = validation_datagen.flow_from_directory(
    "/content/corpus/validation/",
    batch_size=BATCH_SIZE,
    target_size=(WIDTH, HEIGHT),
    shuffle=True)
validation_generator.reset()
show_generate_images(validation_generator)
```

```
accuracy:  74 / 74  =  1.0
```

En nuestro caso el modelo entrenado ha conseguido validar las muestras a la perfección, por lo que no necesitaremos optimizar ningún parámetro de entrenamiento..

Ahora llega el momento de testear el modelo. Lo que haremos será descargar el modelo del colab y crearemos un archivo python en local donde importamos el modelo y unas imágenes sacadas a productos similares para testear. El programa desarrollado y comentado por nosotros es el siguiente:

```
BATCH_SIZE = 100
WIDTH = 250
HEIGHT = 250

new_model = tf.keras.models.load_model("rito_model.h5")  # cargamos el moelo
new_model.summary()  # lo printeamos para ver que sus componentes son correctas

test_datagen = ImageDataGenerator(  # generamos la data de las imagenes
    rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    "./fotos",  # IMPORTANTE: tiene que haber un subdirectorio dentro del directorio escogido
    batch_size=BATCH_SIZE,
    target_size=(WIDTH, HEIGHT),
    color_mode="rgb",
    class_mode=None,  # no definimos ninguna condicion de clase ya que estamos testeando
    shuffle=False)
test_generator.reset()  # importante resetear tras cada llamada

pred = new_model.predict(  # llamamos al metodo predict para predecir la clase de la imagen
    test_generator, verbose=1)

# obtenemos los indices de las clases que hayamos entrenado
predicted_class_indices = np.argmax(pred, axis=1)
# en nuestro caso tenemos 3 clases (0,1,2)

# diccionario donde asociamos el indice de las clases a su nombre correspondiente
labels = {'naranja': 0, 'pasta': 1, 'pera': 2}
labels = dict((v, k) for k, v in labels.items())
# obtenemos el nombre de la clase en vez del indice
predictions = [labels[k] for k in predicted_class_indices]

print("predictions:"+str(predictions))  # comprobamos los resultados
```

En resumen: importamos el modelo y creamos una función para obtener las imágenes de un directorio y predecir sus clases, definidas y asociadas con un diccionario. Si por ejemplo tenemos una pera y una naranja, nos devolverá en forma de lista ordenada su clase correspondiente:

```
predictions:['naranja', 'pera']
```

Ahora solo quedaría implementar el modelo y el archivo py en ROS y testear con las imágenes que recoja el topic de la cámara del robot.

- **Entrenamiento con AWS rekognition**

En vista de los problemas obtenidos al intentar implementar en ROS el modelo entrenado con tensor flow y keras (incompatibilidad de la librería con máquinas virtuales, es

decir, falta de GPU) nos hemos visto obligados a buscar alternativas, y una de ellas fue AWS Rekognition.

Lo primero es iniciar sesión en la consola AWS y situarse en el servicio Rekognition. En el siguiente enlace encontrareis un video guía con todos los pasos del entrenamiento: https://youtu.be/Kped_fesjjU

Una vez terminado con el vídeo, cogemos el código de la API para python y ponemos en la función main nuestro bucket y la imagen que queremos analizar dentro de él. Luego basta con ejecutar el código y obtenemos el label y la probabilidad de acertar:

```python
import re
import boto3
import io
from PIL import Image, ImageDraw, ExifTags, ImageColor, ImageFont
def display_image(bucket, photo, response):
    # Load image from S3 bucket
    s3_connection = boto3.resource('s3')

    s3_object = s3_connection.Object(bucket, photo)
    s3_response = s3_object.get()

    stream = io.BytesIO(s3_response['Body'].read())
    image = Image.open(stream)

    # Ready image to draw bounding boxes on it.
    imgWidth, imgHeight = image.size
    draw = ImageDraw.Draw(image)

    # calculate and display bounding boxes for each detected custom label
    print('Detected custom labels for ' + photo)
    for customLabel in response['CustomLabels']:
        print('Label ' + str(customLabel['Name']))
        print('Confidence ' + str(customLabel['Confidence']))

def show_custom_labels(model, bucket, photo, min_confidence):
    client = boto3.client('rekognition')

    # Call DetectCustomLabels
    response = client.detect_custom_labels(Image={'S3Object': {'Bucket': bucket, 'Name': photo}},
                                           MinConfidence=min_confidence,
                                           ProjectVersionArn=model)

    # For object detection use case, uncomment below code to display image.
    display_image(bucket, photo, response)

    return len(response['CustomLabels'])

def main():

    bucket = 'reko-marc'
    photo = 'naranja.png'
    model = 'arn:aws:rekognition:us-east-1:287394289617:project/rito/version/rito.2021-06-17T18.06.54/1623946015750'
    min_confidence = 95

    label_count = show_custom_labels(model, bucket, photo, min_confidence)
    print("Custom labels detected: " + str(label_count))

if __name__ == "__main__":
    main()
```

```
Detected custom labels for naranja.png
Label naranja
Confidence 95.16799926757812
Custom labels detected: 1
```

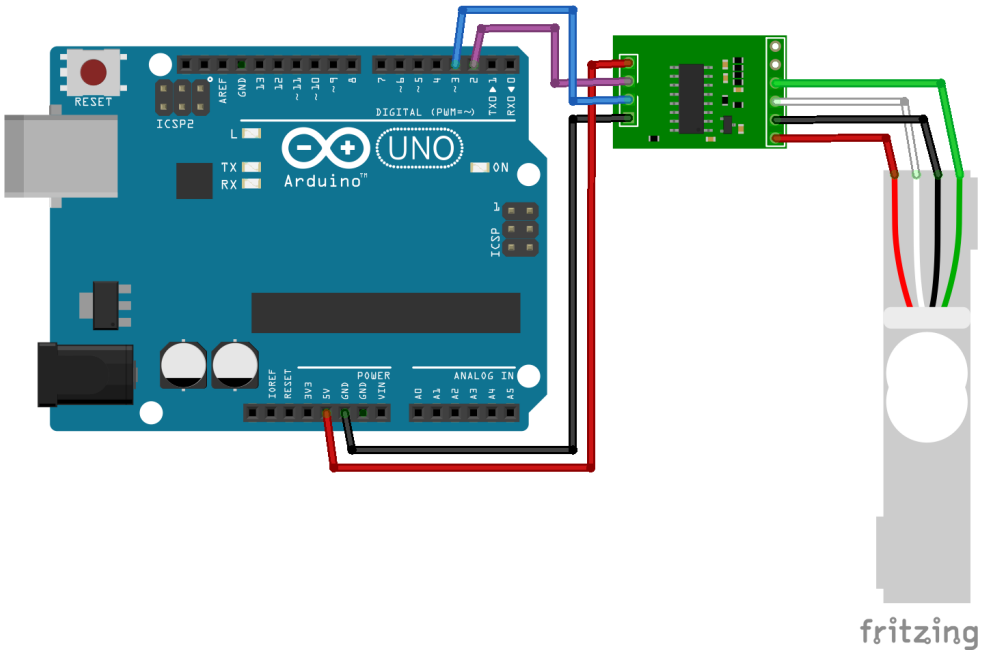
Funcionalidades extra

- **Open CR**

Como se ha comentado anteriormente, también se ha añadido un módulo que nos permite obtener el peso del carrito. Esto se hace gracias a una implementación de galgas mediante Arduino, donde se ha modificado el código de la OpenCR del robot.

Los elementos añadidos son, una galga extensiométrica junto a un microcontrolador HX711, que obtiene los valores analógicos que devuelve la galga extensiométrica y los convierte en digital. Entonces una vez el robot se enciende, calibramos los valores para establecer a 0 la galga extensiométrica y se toman los valores.

La conexión entre estos dos dispositivos son los siguientes:



Como se muestra en la imagen, realizada por fritzing, la galga y el hx711, estan conectados por los cables rojo (E+), negro (E-), blanco (A-) y verde (A+)

El módulo HX711 está conectado a la placa OpenCR del robot, mediante los pines:

HX711	**OpenCR**	**Color Cable**
VCC	5V	Rojo
DAT	Pin 2	Morado
CLK	Pin -3	Azul
GND	GND	Negro

Simplemente, se ha modificado el código original de la OpenCR y se ha añadido el módulo de obtener el peso, de la siguiente forma:

Se han definido los pines 2 y 3 para sck y dout respectivamente

```
#define LOADCELL_DOUT_PIN 3
#define LOADCELL_SCK_PIN 2
```

Se inicia el microprocesador introduciendo los pines anteriores para obtener los valores, estableciendo una escala de 2280, porque era la más correcta para poder obtener valores más cerca de lo correcto.

```
scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);

scale.set_scale(2280.f);                    // th
scale.tare();
```

Se ha creado una función getWeight para obtener los valores del objeto scale, que obtiene los resultados continuamente de la galga extensiométrica. Y se ha utilizado un un get units con 10, para que se obtenga una media de los 10 valores, para obtener un valor más preciso.

```
float getWeight() {
  float weight;
  weight = scale.get_units(10);
  return weight;
}
```

Por último, se ha creado una función publishWeightMsg, en la que establecemos los valores para establecer el mensaje y publicarlo.

```
void publishWeightMsg(void) {
  range_msg.radiation_type = sensor_msgs::Range::ULTRASOUND;
  range_msg.header.frame_id =  frameid;
  range_msg.field_of_view = 0.1;
  range_msg.min_range = 0.0;
  range_msg.max_range = 6.47;
  range_msg.range = getWeight();
  range_msg.header.stamp = nh.now();
  pub_range.publish(&range_msg);

}
```

```
if ((t - tTime[7]) >= (1000 / ULTRASOUND_PUBLISH_FREQUENCY))
{
  publishWeightMsg();
  tTime[7] = t;
}
```

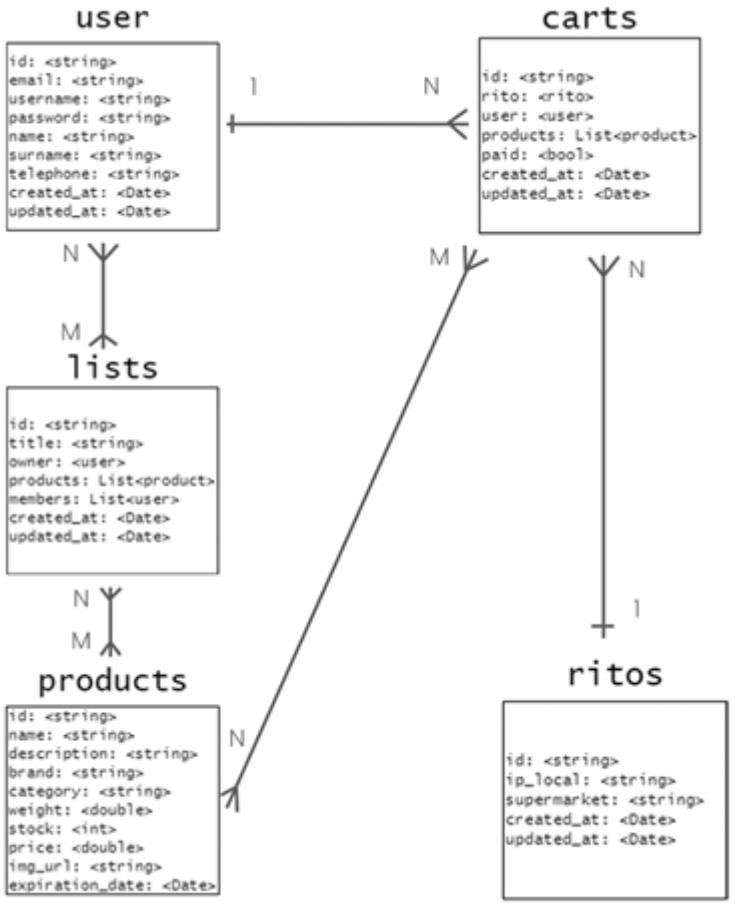
Una vez se empieza a publicar estos valores dentro del servicio, el robot es capaz de obtener esa información y de esta forma podemos mostrar estos valores redondeados en la App Web y permitir que el usuario obtenga esa información de manera más sencilla y de forma intuitiva.

- **Base de datos**

Con el fin de almacenar la información referente al usuario (sus datos, sus listas, etc.) y almacenar los datos referentes al supermercado (productos, carros de la compra, etc.) decidimos utilizar una base de datos.

Utilizamos una base de datos no-relacional MongoDB. Nos decidimos por esta tecnología por dos motivos. Primero porque nunca habíamos usado una base de datos no-relacional que no fuera Firebase. El otro motivo fue porque necesitábamos disponer de la base de datos rápidamente y que estuviera abierta a cambios (no tan estricta como las bases de datos relacionales como las que hemos utilizado en otros proyectos).

Las relaciones y entidades de nuestra base de datos son las siguientes:



User: Usuarios de la aplicación y sus datos personales.
Lists: Listas de la compra que crean los usuarios. Un usuario puede compartir sus listas con otros usuarios.
Products: Productos del supermercado y sus datos.
Ritos: Carros de compra – Robots. La parte hardware del proyecto.
Carts: Objeto que representa la cesta de la compra.

- **Servidor**

Para disponer de los datos anteriormente mencionados hemos utilizado un servidor. Para el desarrollo del proyecto teníamos claro que el producto no podía ser viable si el usuario no podía darse cuenta de que la compra que estaba realizando estaba correctamente en todo momento. Así pues, la importancia residía en mostrar los cambios en la posición del carro, en la cesta y la ruta y los productos si se añadía uno durante la compra en tiempo real. Por este motivo decidimos usar Feathers.

Feathers es un framework web orientado a servicios y a la comunicación en tiempo real. Feathers permite que tanto servidor como clientes envíen y reciban información

mediante websockets y si es necesario, enviar un evento a los demás clientes interesados en esta información. Está montado por encima de Express y como éste, también tiene a su disposición pequeños módulos para requerimientos distintos.

Para hacer uso de esta comunicación en tiempo real, solamente tenemos que crear un servicio en el servidor y exponerlo a través de una interfaz REST.

En nuestro caso, estos servicios están asociados con los datos de la base de datos. Afortunadamente, Feathers cuenta con varios adaptadores para bases de datos MongoDB. Nosotros nos hemos decantado por Mongoose, ya que la implementación es rápida y permite utilizar esquemas de las entidades y crear algo parecido a las claves foráneas de las bases de datos relacionales SQL.

Para la autenticación se ha usado un servicio de feathers expuesto a través de una interfaz REST que, si los datos del inicio de sesión son correctos, devuelve un JWT que se guarda de forma local en el navegador del cliente.

También hemos hecho uso de los Hooks de Feathers, que son funciones que se ejecutan antes o después de ejecutar la petición al servidor. Por ejemplo, cuando al servidor le llega una petición de obtener las listas de la compra, antes de hacer la consulta se hace uso del Hook de autenticación que se asegura de que el cliente que la pide es un usuario válido y con la sesión abierta. Después de hacer la consulta en la base de datos, analizamos estas listas para solamente responder con aquellas de las que el usuario es dueño o miembro.

Reflexión crítica

En cuanto al desarrollo del proyecto ha sido algo más duro de lo que nos esperábamos, nos han surgido más problemas específicos que no estamos acostumbrados a resolver. Nuestro mayor problema ha sido la organización, y también gran parte la falta de motivación. Ha sido un entorno difícil de controlar, junto a las máquinas virtuales y Gazebo, de este modo, empezamos el proyecto un poco desganados. Las asignaturas en general de todo el curso, no han sido las que más nos han gustado, exceptuando IA. Entonces, al final acaba influyendo en nuestras ganas. Ha sido un proyecto muy interesante con muchas cosas con las que se pueden investigar y sacar un mayor potencial.

Si nosotros como equipo, nos hubiésemos organizado de una mejor forma, estamos seguros de que hubiese ido mejor. Empezamos con una muy buena idea de proyecto, que se nos fue desinflando cada vez más.

Aunque no acabemos con la mayor nota, creo que hemos aprendido bastante y sobre todo en ser autocríticos con nosotros mismos y con nuestro equipo, criticando y dando

opinión de todo aquello que no nos acababa de gustar. Pero siempre desde el respeto. Creo que como equipo, tenemos una gran amistad, que nos ha ayudado a no dejar más de lado el proyecto y recuperar en los últimos sprints.

Conclusiones

En líneas generales y a nivel de grupo podría decirse que no ha sido el proyecto que más nos ha entusiasmado, cosa que no quita que hayamos aprendido tecnologías diferentes como es ROS o Python. La parte que quizás nos haya llamado más la atención ha sido el reconocimiento y la máquina de estados, sobre todo por el potencial y el futuro que tienen. Este proyecto probablemente sea el más demandado de la carrera a nivel laboral ya que mezcla una gran abanico de tecnologías diferentes que en unos años formarán parte de nuestro día a día. Todos estamos de acuerdo que la parte de hardware es la que menos nos llama la atención, pero al mismo tiempo es una parte imprescindible para el correcto desarrollo del proyecto.

Como mejora ya comentamos el actualizar las versiones de ROS y Gazebo en cuanto a herramientas. Referente al profesorado, todo perfecto, buen seguimiento a lo largo de los sprints y atentos a las dudas que nos surgían.

Historial de Versiones

Versión	Autor	Descripción	Fecha
v1.0	Marc	Creación de la estructura del documento	21/06/2021
v1.1	Fabio Hernández	Introducción	23/06/2021
v1.2	Ignasi Mari	Mock-Up del diseño web Explicación de las tecnologías web Esquema general de la aplicación Explicación Base de Datos	23/06/2021
v1.3	David Jiménez	Explicación Servidor	23/06/2021
v1.4	Sergio Pujol	Objetivos y funcionalidades del proyecto	23/06/2021

v1.5	David Jiménez	Revisión Documento & Corrección de errores. Corrección del Mock-Up	23/06/2021
v1.6	Marc Oller	Reconocimiento con OpenCV Máquinas de estados	23/06/2021
v1.7	Fabio Hernández	Reconocimiento TensorFlow Reconocimiento Rekognition	24/06/2021
v1.8	Sergio Pujol	Funcionalidad Extra: Open CR	24/06/2021
v1.9	Ignasi Marí	Corrección Diagrama de Flujo del diseño web	25/06/2021
v2.0	Marc Oller	Reestructuración del informe	25/06/2021