TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

Integración de un robot humanoide anfitrión en el laboratorio Smarthome.

Curso: 2020/2021

Alumno/a:

Blanco Álvarez, Aarón

Director/es:

SanJuan Estrada, Juan Francisco

Texto de la dedicatoria

ÍNDICE GENERAL

Página

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTADOS

ABREVIATURAS

ESI	Escuela Superior de ingeniería
InSo	Ingeniería del Software
Tic	Teconología de la información
TFG	Trabajo fin de grado
CTFG	Complemento de trabajo de fin de grado
UAL	Universidad de Almería
SLAM	Simultaneous Localization and Mapping
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
PNG	Portable Network Graphics
SSH	Secure Shell

RESUMEN Y ABSTRACT

En este archivo se incluirá tanto el resumen en castellano como su traducción al inglés. Los dos párrafos estarán ligeramente separados.

El propósito del trabajo en una o dos frases. El diseño y metodología utilizada, los resultados más significativos del trabajo realizado y un breve resumen de las conclusiones. Debe ser conciso y presentar los resultados obtenidos tras la ejecución del TFG.

Put here the english tranlation ... . Debe ser conciso y presentar los resultados obtenidos tras la ejecución del TFG. Los dos parrafos estarán ligeramente separados.

1 Introducción

INTRODUCCIÓN A HACER AL FINALIZAR EL TRABAJO.

2 Consideraciones generales

2.1 Objetivos

El principal objetivo del TFG, será el desarrollo de una aplicación para el robot Pepper. La finalidad de esta aplicación va a ser la de dotar al robot la capacidad de resolver dudas que se le plantee, para ello, se van a desarrollar habilidades que le permitirá interactuar. Las habilidades desarrolladas, se centrarán enseñar al robot para actuar como anfitrión del laboratorio, de tal forma que sea capaz de enseñar y explicar las instalaciones del laboratorio, sus funcionalidades e incluso los experimentos desarrollados por los estudiantes de Máster, incluyendo gestos naturales de una persona y usando un lenguaje coloquial para amenizar la visita. Además, adquirirá la capacidad para resolver algunas de las posibles dudas sobre el laboratorio Smarthome, permitiendo una interlocución entre los visitantes y el robot. Para lograr los diversos objetivos planteados se requerirá:

• Estudio del entorno de desarrollo del robot que viene preparado para el despliegue de aplicaciones.

• Recopilación de la información sobre el laboratorio Smarthome y los distintos dispositivos y servicios que ofrece para que puedan ser integrados en las explicaciones que dará el robot.

• Se va a estudiar la implementación de contenido que reproducirá el robot usando un proveedor de servicios de nube como puede ser (Amazon, Microsoft, Google, IBM. . .) para poder resolver las cuestiones planteadas de forma más eficaz.

• Se estudiará la capacidad de dar movimiento al robot teniendo en cuenta los distintos sensores de movimiento junto a visuales que tiene para poder enseñar de forma más coherente los elementos en el laboratorio.

• Se pretende dotar de gestos naturales que permitan expresar al robot un lenguaje más natural. A continuación, se hará una ampliación al TFG, con un Complemento del Trabajo de Fin de Grado, (CTFG) que aumentará la capacidad del robot adaptar su lenguaje a la visita guiada para distintos perfiles de visitantes, diferenciando entre edades (adultos, jóvenes o niños), discapacidades (auditiva, visual, intelectual).

2.2 Historia de la robótica

Desde el origen de las civilizaciones, el hombre siempre ha mostrado un gran afán por construir máquinas a su imagen y semejanza. Existen referencias ya en la China clásica (año 500 a.C.) de la construcción de una urraca voladora de madera y bambú y de un caballo

de madera capaz de dar saltos. En la antigua Grecia, Arquitas de Tarento (hacia el año 400 a.C.) construyó un autómata que consistía en una paloma de madera que rotaba por sí sola gracias a un surtidor de agua o vapor y simulaba el vuelo. En el Renacimiento, Leonardo da Vinci construyó para el rey Luis XII de Francia un león mecánico que se abría el pecho con la garra y mostraba los símbolos del escudo de armas real; también diseñó alrededor del año 1495 uno de los primeros mecanismos con forma completamente humana, vestido con una armadura medieval, aunque, como muchos otros de sus inventos, no se llegó a construir. Este mecanismo se ha reconstruido en la actualidad según los dibujos originales, y puede mover los brazos, girar la cabeza y sentarse. En la catedral de Estrasburgo funcionó desde 1352 hasta 1789 un gallo mecánico, que formaba parte del reloj, que movía el pico y las alas al dar las horas.[4]

2.3 Robot Pepper

El robot Pepper ha sido diseñado por la antigua Aldebaran Robotics, hoy en día SoftBank Robotics empresa mundialmente conocida sobre todo por la reciente adquisición de Boston dynamic y hasta el año 2020, propietaria de ARM. Pepper fue inicialmente diseñado para estar dedicado a las empresas para dar un servicio a sus consumidores. En aquel momento en Junio de 2014 ya existían distintas plataformas, sin embargo a diferencia de otras que estaban dedicados a negocios, finalmente Pepper pudo ser reutilizado para cualquier tipo de consumidor.[5]

2.3.1 Especificaciones técnicas

Las especificaciones técnicas del robot más relevantes son las siguientes:

1. Peso: 28 kg
2. Altura: 120 cm
3. Fondo: 42,5 cm
4. Batería: Litio, 30,0Ah/795Wh
5. Autonomía: Hasta 12 horas.
6. Conectividad: Wi-Fi / Ethernet
7. Velocidad: Más de 3km/h
8. Motores: 20
9. Partes móviles: Cabeza (1), hombros (2),
   codos (2), muñecas (2), dedos (10), caderas (1) y rodillas (1)
10. Ruedas: 3 (omnidireccionales)
11. Movimiento: 360º
12. Velocidad máxima: 3 km/h

13. Tablet: LG CNS

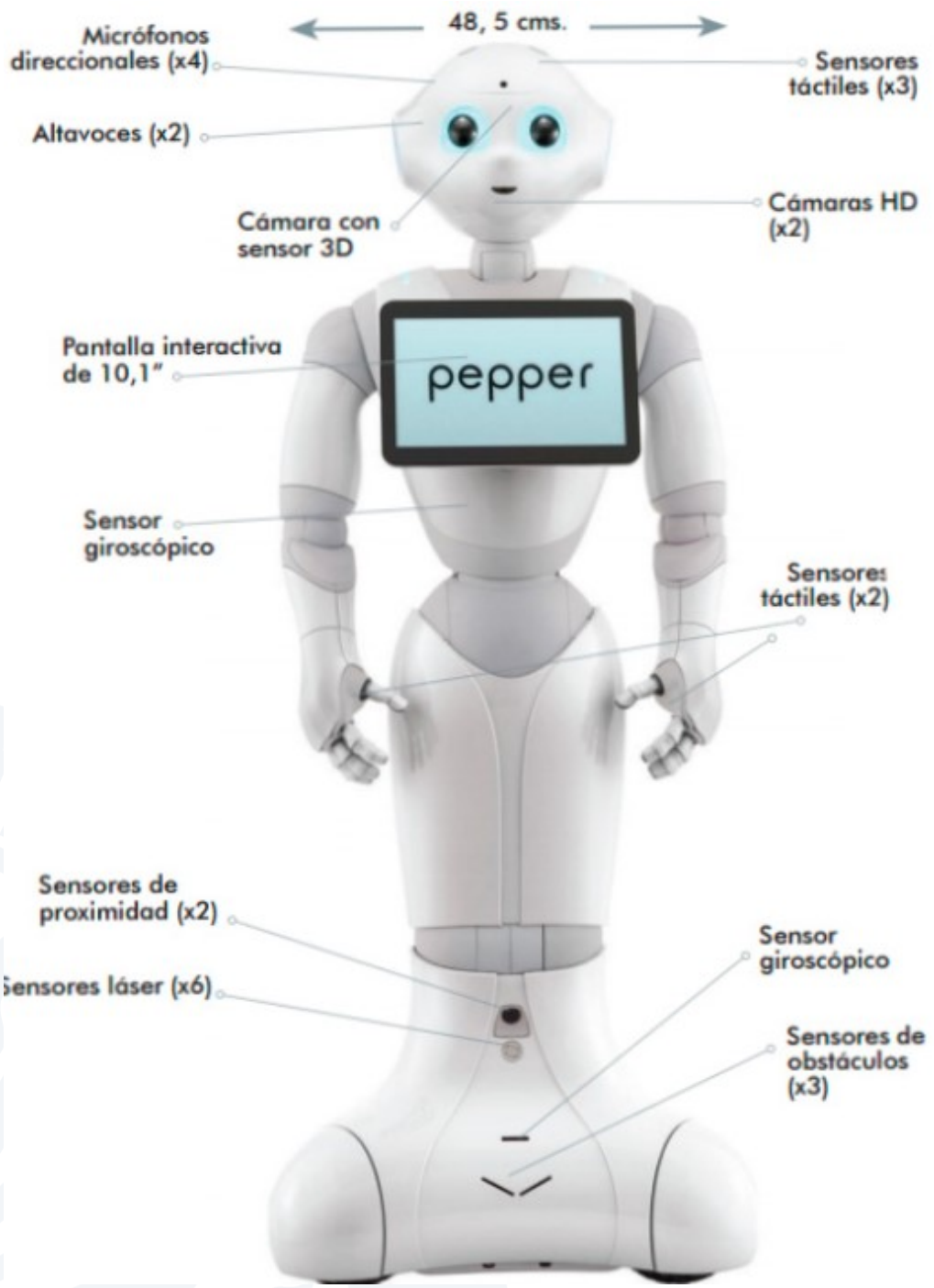
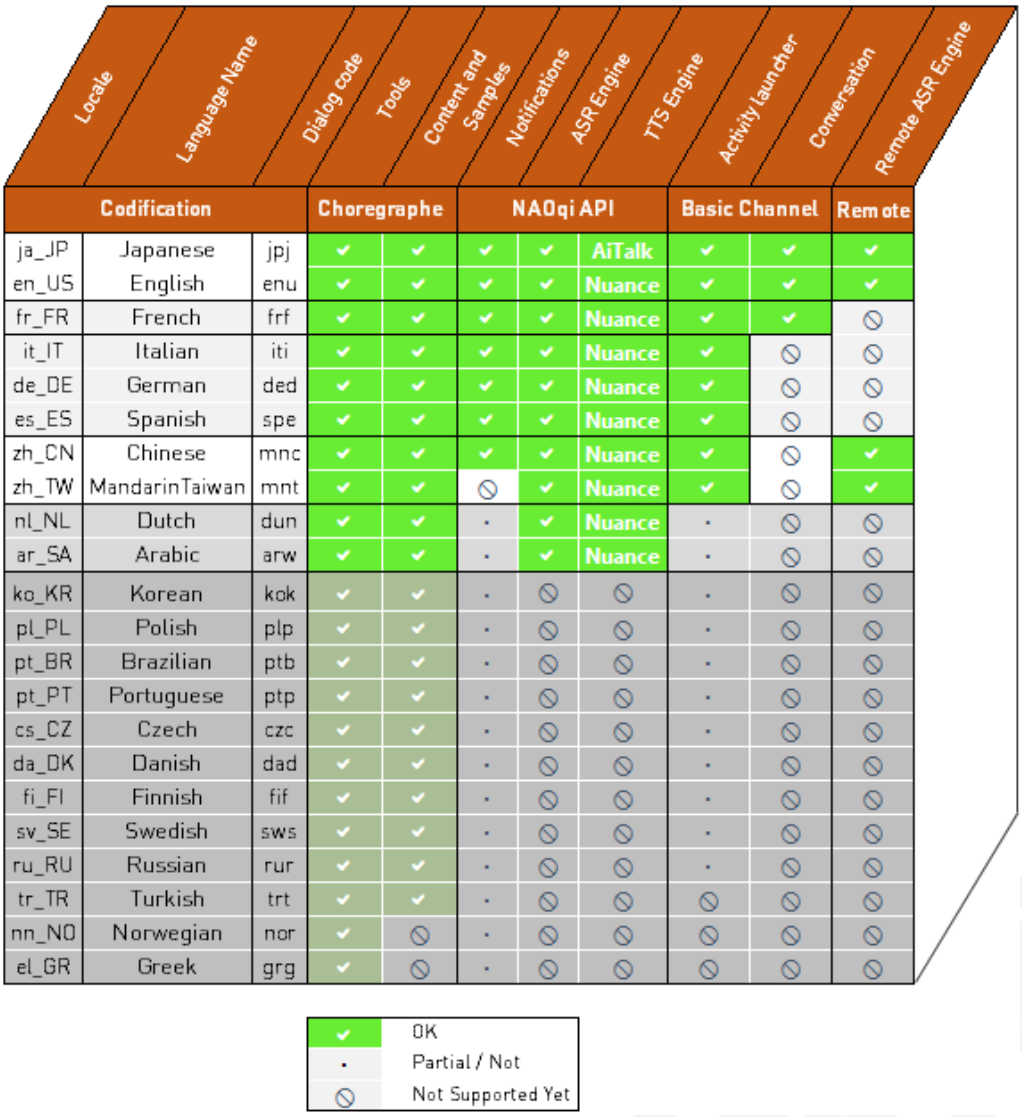


Figura 2.1: Robot Pepper. [1]

Pepper viene en idioma español, sin embargo, esto trae consigo algunos inconvenientes a diferencia de las versiones inglesa y japonesa. Veamos una gráfica que la empresa fabricante nos provee con los idiomas soportados:



Locale	Language Name	Dialog code	Tools	Content and Samples	Notifications	ASR Engine	TTS Engine	Activity launcher	Conversation	Remote ASR Engine
Codification			Choregraphe		NAOqi API			Basic Channel		Remote
ja_JP	Japanese	jpj	✓	✓	✓	✓	AiTalk	✓	✓	✓
en_US	English	enu	✓	✓	✓	✓	Nuance	✓	✓	✓
fr_FR	French	frf	✓	✓	✓	✓	Nuance	✓	✓	⊘
it_IT	Italian	iti	✓	✓	✓	✓	Nuance	✓	⊘	⊘
de_DE	German	ded	✓	✓	✓	✓	Nuance	✓	⊘	⊘
es_ES	Spanish	spe	✓	✓	✓	✓	Nuance	✓	⊘	⊘
zh_CN	Chinese	mnc	✓	✓	✓	✓	Nuance	✓	⊘	✓
zh_TW	MandarinTaiwan	mnt	✓	✓	⊘	✓	Nuance	✓	⊘	✓
nl_NL	Dutch	dun	✓	✓	·	✓	Nuance	·	⊘	⊘
ar_SA	Arabic	arw	✓	✓	·	✓	Nuance	·	⊘	⊘
ko_KR	Korean	kok	✓	✓	·	⊘	⊘	·	⊘	⊘
pl_PL	Polish	plp	✓	✓	·	⊘	⊘	·	⊘	⊘
pt_BR	Brazilian	ptb	✓	✓	·	⊘	⊘	·	⊘	⊘
pt_PT	Portuguese	ptp	✓	✓	·	⊘	⊘	·	⊘	⊘
cs_CZ	Czech	czc	✓	✓	·	⊘	⊘	·	⊘	⊘
da_DK	Danish	dad	✓	✓	·	⊘	⊘	·	⊘	⊘
fi_FI	Finnish	fif	✓	✓	·	⊘	⊘	·	⊘	⊘
sv_SE	Swedish	sws	✓	✓	·	⊘	⊘	·	⊘	⊘
ru_RU	Russian	rur	✓	✓	·	⊘	⊘	·	⊘	⊘
tr_TR	Turkish	trt	✓	✓	·	⊘	⊘	⊘	⊘	⊘
nn_NO	Norwegian	nor	✓	⊘	·	⊘	⊘	⊘	⊘	⊘
el_GR	Greek	grg	✓	⊘	·	⊘	⊘	⊘	⊘	⊘

Figura 2.2: Lenguajes soportados por Pepper. [2]

Como podemos observar, existen módulos con los que nuestro robot no cuenta como puede ser el módulo de conversación qué nos otorgaría la capacidad de hablar con el robot más allá de responder ordenes básicas.

2.3.2 NAOqi 2.5

NAOqi es el el principal software que es utilizado por el robot y lo controla. Este framework es usado para programar los robots Nao. Algo destacable de este software es que está hecho para ser multiplataforma. [6]

2.4 HERRAMIENTAS

A la hora de realizar este proyecto, se han usado distintas herramientas.

2.4.1 Choreographe

Choreographe es un entorno de programación para facilitar la programación de sistemas NAO. Su funcionamiento es mediante diagramas de flujo encadenando cajas qué escritas en código Python. Tiene diversas herramientas qué permiten la conexión con el robot pepper, esto incluye un monitor de vídeo donde podemos ver a qué está enfocando el robot. Otra de las ventajas de esta herramienta es la virtualización del robot que pese a ser limitada, nos permite probar comportamientos en caso de no tener un robot a mano. Las cajas con las que programamos los comportamientos, son accesibles, pudiendo encontrar el código en Python de cada una, además podemos crear nuestros propios scripts permitiendo integrar la funcionalidad que veamos.[2]

2.4.2 Python

Lenguaje en auge orientado a objetos y soportado por la comunidad, permite acceder a pepper y con el, ejecutar scripts o mediante el interprete, acceder a las librerías de NAOqi y poder ejecutar las funciones deseadas. Python permite construir scripts propios donde podemos ir poniendo las instrucciones que queramos ejecutar dentro de Pepper. Las ventajas que tiene este lenguaje es que tiene fácil legibilidad junto a una corta curva de aprendizaje. Además, es el lenguaje base con el que funcionan las cajas de Choreographe, permitiendo fácilmente el uso de scripts en la herramienta.[2]

2.4.3 C++

Este lenguaje es el más común usado en robótica, este permite crear todo tipo de nuevas funcionalidades accediendo a la información de los sensores en tiempo real. Lo interesante de este lenguaje es su compatibilidad con ROS, un meta-sistema operativo enfocado a la robótica. Además existe una gran cantidad de herramientas compatibles con C++, sin embargo, estamos hablando de un lenguaje de programación compilado y más complejo.[2]

3 Uso de módulos del robot

3.1 Consideraciones generales

A la hora de crear una aplicación en Choreographe, tenemos que modificar diversas propiedades ya que tendremos fallos si no tiene la configuración correcta. El principal problema que nos podemos encontrar es el lenguaje. El robot Pepper soporta diversos lenguajes como hemos podido ver antes en la figura 2.2. Sin embargo, cuando se crea una aplicación, hay que establecer

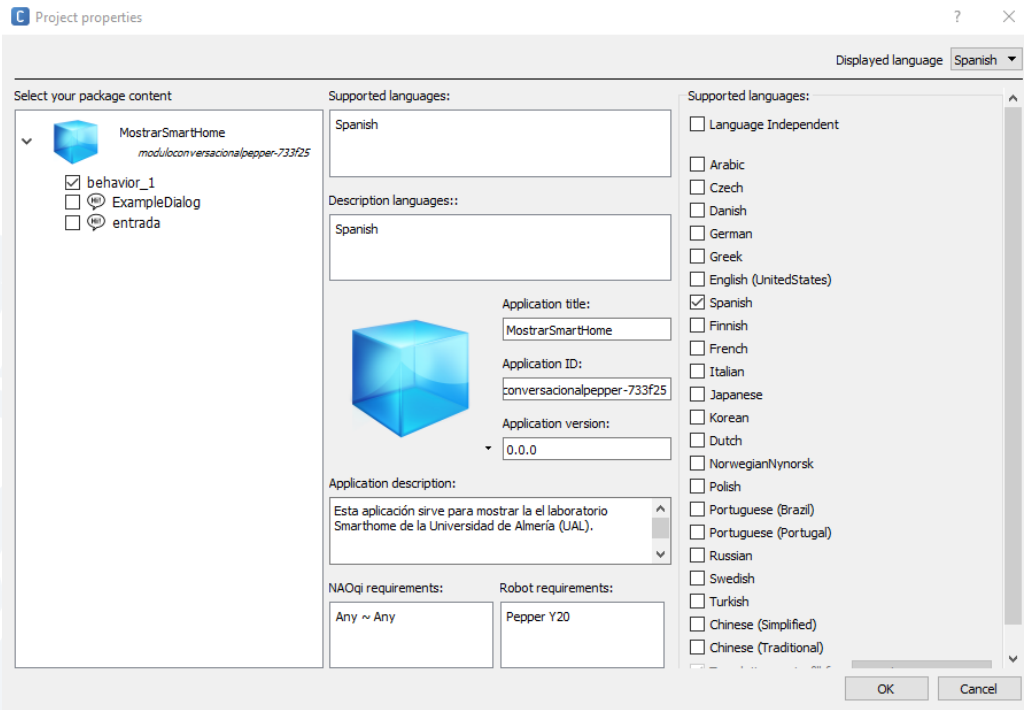


Figura 3.1: Propiedades del proyecto.

3.2 Exploración

A la hora de plantear la presentación, se plantea el uso de los distintos sensores del robot de manera en la que se pueda generar un mapa del laboratorio. Para generar movimiento controlado se plantea el uso de SLAM[7]. Para la generación de un mapa, se hace uso de un script qué usa la librería **ALNavigation**, aquí podemos hacer uso de diversas funciones como:

- *explore*, esta función es la qué se ocupa de realizar la exploración. Aquí se introduce por parámetro el radio que va a realizar el robot.

```
# Explore the environement, in a radius of 10 m.
    radius = 10.0
    error_code = navigation_service.explore(radius)
    if error_code != 0:
        print "Exploration failed."
        return
```

Listado 3.1: Instrucciones para comenzar a explorar.

- *saveExploration*, esta función nos permite definir la ruta donde guardaremos el mapa.

```

    # Saves the exploration on disk
    path = navigation_service.saveExploration()
    print "Exploration saved at path: \"" + path + "\""
    # Start localization to navigate in map
```

Listado 3.2: Instrucciones para guardar el mapa.

- *startLocalization*, con esta llamada, el robot empezará a ubicarse dentro del mapa gracias a sus sensores.

```
   # Stop localization
    navigation_service.stopLocalization()
```

Listado 3.3: Intrucciones para parar de localizarse.

- *NavigateToInMap*, esto permite al robot moverse a las posiciones indicadas en relación al mapa.

```
   # Come back to initial position
    navigation_service.navigateToInMap([0., 0., 0.])
```

Listado 3.4: Instruccioens para navegar hacia el lugar de inicio.

Estas funciones son las que nos permitirán generar un mapa del laboratorio codificado con la extensión .explo, tal y como podemos ver en la figura 3.2 aparentemente es difícil de entender debido a su codificación, sin embargo, podemos aplicar otras funciones para recrear el mapa en 2D como veremos más adelante en la figura 3.3.

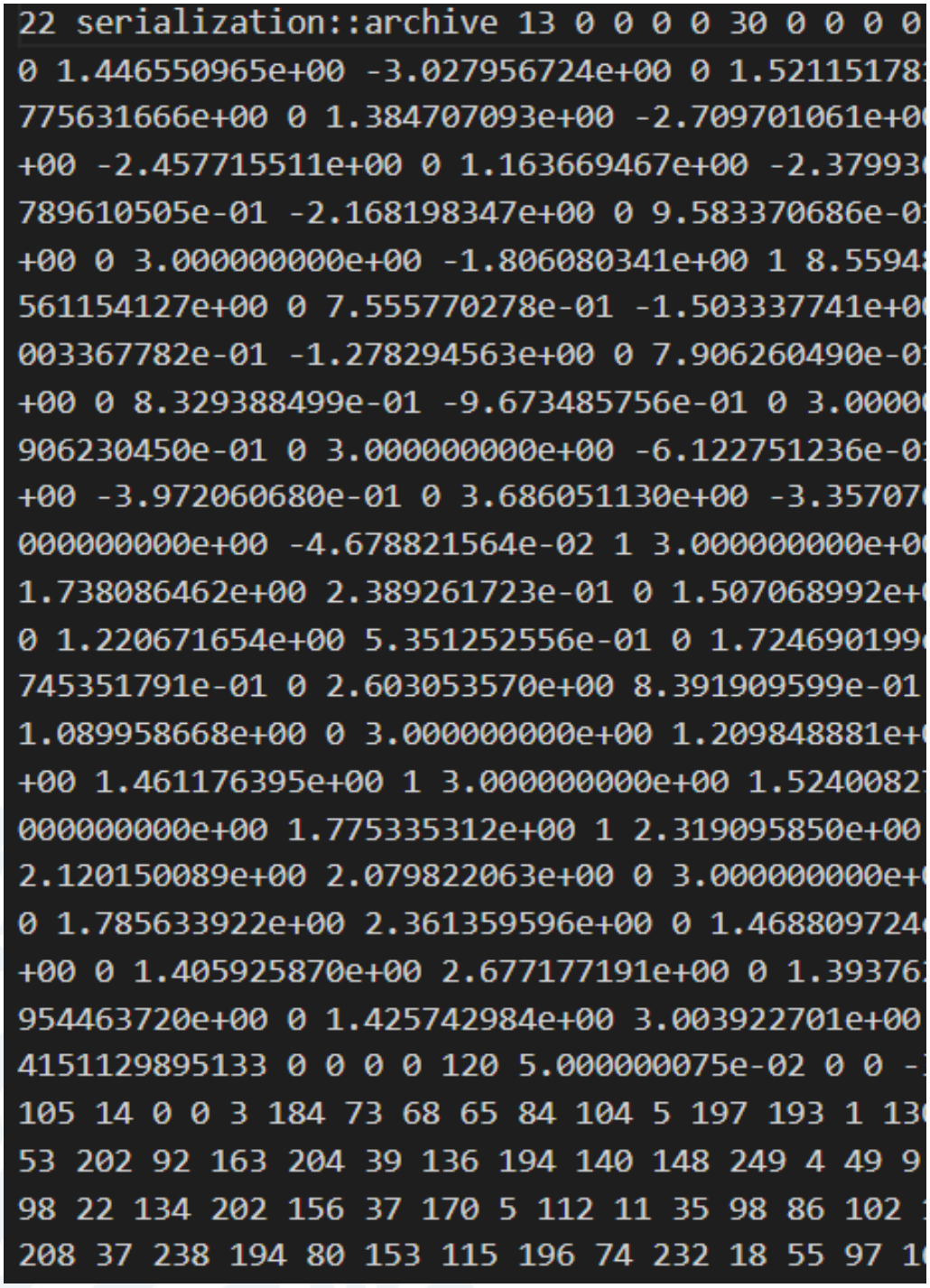


Figura 3.2: Archivo con extensión EXPLO.

Para realizar este mapa hemos usado el siguiente script que tenemos a continuación:

```
1
2 #! /usr/bin/env python
3 # -*- encoding: UTF-8 -*-
4
5 """Example: Use explore method."""
6
```

```
import qi
import argparse
import sys
import numpy
import Image


def main(session):
    """
    This example uses the explore method.
    """
    # Get the services ALNavigation and ALMotion.
    navigation_service = session.service("ALNavigation")
    motion_service = session.service("ALMotion")

    # Wake up robot
    motion_service.wakeUp()

    # Explore the environement, in a radius of 10 m.
    radius = 10.0
    error_code = navigation_service.explore(radius)
    if error_code != 0:
        print "Exploration failed."
        return
    # Saves the exploration on disk
    path = navigation_service.saveExploration()
    print "Exploration saved at path: \"" + path + "\""
    # Start localization to navigate in map
    navigation_service.startLocalization()
    # Come back to initial position
    navigation_service.navigateToInMap([0., 0., 0.])
    # Stop localization
    navigation_service.stopLocalization()
    # Retrieve and display the map built by the robot
    result_map = navigation_service.getMetricalMap()
    map_width = result_map[1]
    map_height = result_map[2]
    img = numpy.array(result_map[4]).reshape(map_width, map_height)
    img = (100 - img) * 2.55 # from 0..100 to 255..0
    img = numpy.array(img, numpy.uint8)
    Image.frombuffer('L',  (map_width, map_height), img, 'raw', 'L', 0, 1).show()
```

Listado 3.5: Script completo generador de mapa en formato .explo

Este script incluye la conexión al robot mediante el inicializador, adicionalmente, y tras el uso de algunas funciones que hemos descrito antes, en este caso, tras terminar la exploración pediremos al robot qué vuelva a la posición de inicio. A la hora de realizar el mapa, nos apoyaremos en la función **getMetricalMap** esta nos devolverá una serie de colecciones qué incluirán la resolución del mapa por pixel, la anchura y altura, el origen y el buffer que indica el espacio libre y ocupado. Con estos datos junto a funciones de la librería **numpy** podemos transformarlos en el mapa qué hemos visto en la anterior figura 3.2.

Para hacernos una idea sobre el mapa y lo qué el robot ha leído, podemos usar el siguiente script que se apoya en la librería opencv. En este caso, haremos uso de la función anteriormente presentada **getMetricalMap** para leer el mapa, y junto a numpy podemos codificar las colecciones para crear una imagen png. También nos apoyaremos del uso de textbfbase64, con lo que podemos codificar y decodificar imágenes.

```
import base64
import cv2
```

```
import numpy as np

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.mem = ALProxy("ALMemory")
        self.nav = ALProxy("ALNavigation")

    def onLoad(self):
        #put initialization code here
        pass

    def onUnload(self):
        #put clean-up code here
        pass

    def onInput_onStart(self):
        # Get the map from navigation.
        map = self.nav.getMetricalMap()
        # mpp = map[0]
        #print "mpp " + str(mpp)
        size = map[1]
        print "size " + str(size)
        # map[2] is size as well.
        # map[3] is origin offset, not needed for now.
        data = map[4]

        # Fit the size of the image
        img = np.array(data, np.uint8).reshape(size, size, 1)
        img = (100 - img) * 2.5
        shape = img.shape
        img = img.transpose((1, 0, 2)) # Do not transpose the channels.
        shape = img.shape
        # png
        flag, buff = cv2.imencode(".png", img)
        #print "Flag " + str(flag)
        # base 64
        buff64 = base64.b64encode(buff)
        self.logger.warning("")
        full = "" + buff64
        self.logger.warning(full)

        # show app
        self.mem.raiseEvent("Navigation/displayMap", full)

        #activate the output of the box
        self.onStopped()

    def onInput_onStop(self):
        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
        self.onStopped() #activate the output of the box
```

Listado 3.6: Codigo para crear el mapa

Podemos ver el resultado en la figura 3.3, como se puede observar, pepper ha podido crear un mapa que no es absolutamente fiel a la realidad, sin embargo, se puede apreciar las distintas zonas que existen en el laboratorio.

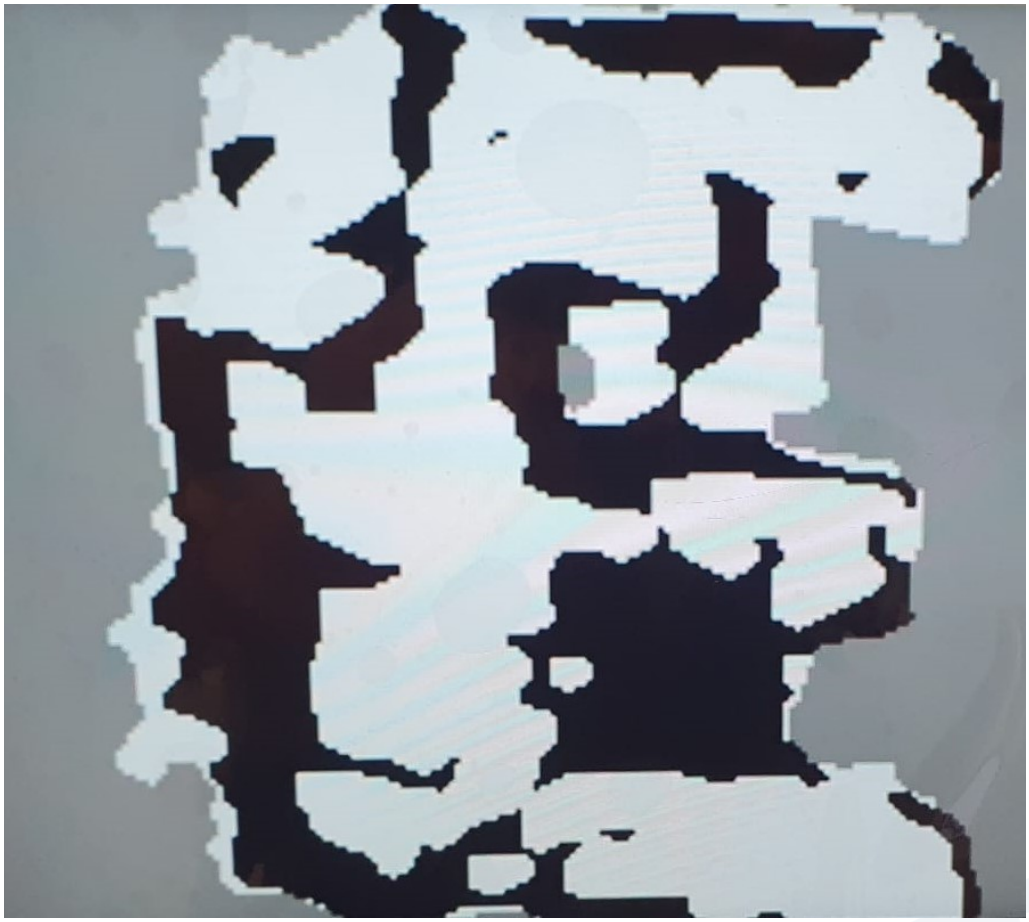


Figura 3.3: Mapa del laboratorio.

3.3 NAVEGACIÓN

Para la navegación, Pepper usa 3 ruedas omnidireccionales que le permiten moverse libremente por un espacio mientras sus sensores no detecten obstáculos. Pepper puede llegar a una velocidad de más de 3 km/h, una velocidad controlable que le permite maniobrar con relativa facilidad pero siempre manteniendo una distancia de seguridad [1]. Para este trabajo, se ha usado dos funciones de las distintas que nos provee la API de NAOqi [8].
Por una parte, Pepper se moverá por el mapa mediante coordenadas generadas en base a la exploración previa, por tanto cargaremos el mapa y junto a el, usaremos la función **NavigateToInMap**, esta función se mostró antes en el listado 3.3 pero en este caso, se especificarán las coordenadas de las distintas posiciones. Desde el lavabo al dormitorio.
Por otra parte, para permitir el giro del robot sobre su eje, hacemos uso de la librería ALMotion [9] qué con la función **moveTo** nos permite establecer el número de grados qué girará.

```
1        float x  = 0.2f;
2        float y  = 0.2f;
3        float theta = 1.5709f;
4        motion.moveTo(x, y, theta);
```

Listado 3.7: Llamada a moveTo.

3.4 Uso de el módulo de voz

A la hora de usar el módulo de voz hay varias cosas que se tienen que tener en cuenta. Como en este caso el robot Pepper viene con una versión en español pero sin embargo por defecto las aplicaciones vienen en inglés, por ello hay que hacer diversos cambios. A parte de un cambio en las propiedades de la aplicación como vimos en la figura 3.1, es necesario también establecer el idioma en la aplicación al principio de toda la ejecución. Esto lo haremos al principio de nuestro panel de la aplicación con la caja **Set Language** como vemos en la figura 3.4, en caso de qué nuestra aplicación soporte distintos idiomas, aquí se podrá cambiar el idioma y salvo qué se establezca otro idioma, se usará el seleccionado.

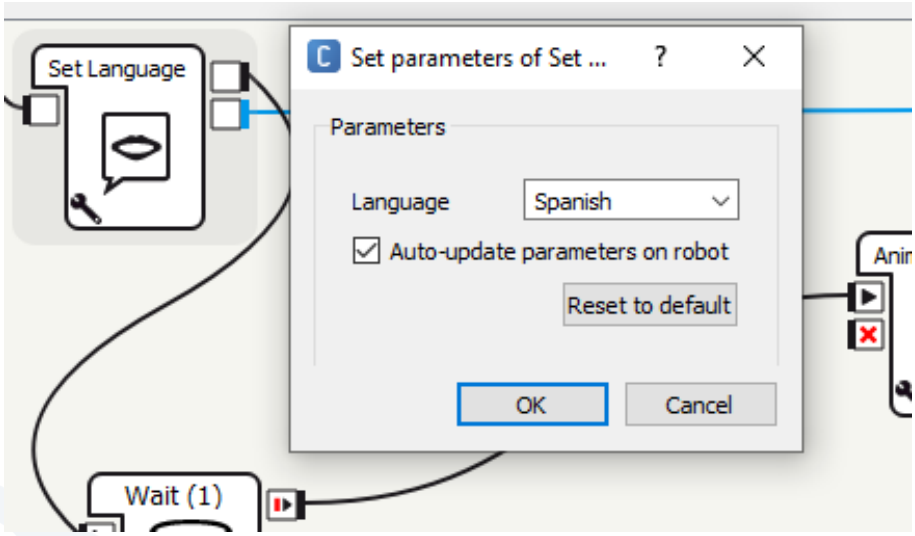


Figura 3.4: Seleccionar idioma.

El objetivo principal del robot es trasmitir una descripción en las distintas zonas del laboratorio. Para ello hay diversas maneras de enfocarlo. NAOqi permite por ejemplo acceder a las librerías directamente a través de **ALTextToSpeech**, esto lo hemos hecho directamente en algunos scripts, por ejemplo en caso de que el robot esté perdido, este hará uso de la librería, avisando de su condición como vemos a continuación.

```
1            sentence = "Estoy perdido."
2            self.tts.post.say(str(sentence))
```

Listado 3.8: Uso de ALTextToSpeech

Sin embargo, a lo largo de la aplicación, vamos a hacer uso de las cajas de texto que usan animaciones para dar contexto cuando el robot se expresa. Esto es porque aunque podamos usar las llamadas directamente en scripts y definir las animaciones manualmente, se pueden hacer con la caja **say animation** qué cargará automáticamente el texto y pondrá animaciones en función del texto que esté leyendo.

3.5 USO DE LA TABLET

El robot viene con una tablet incluida de 246 x 175 x 14’5 cm qué destaca a la altura de su pecho. Esta tablet viene con un salva-pantallas por defecto qué se enciende al encender a Pepper.
Para mejorar la comprensión de las distintas explicaciones, Pepper puede dar un énfasis a ello mediante el uso de su tablet. Este uso de la tablet se puede hacer de distintas maneras, por ejemplo, se puede usar mediante las caja **show webApp**. Utilizamos esto para la aplicación explore con la cual activamos la tablet y ejecutamos la aplicación en JS. Adicionalmente, aquí establecemos las distintas rutas internas a los archivos qué usaremos para mostrar en la tablet mediante el uso de **loadApplication** donde pondremos la ruta a la aplicación.

```
1  tabletService = self._getTabletService()
2        appName = self.packageUid()
3        state = False
4        if appName:
5            if tabletService:
6                if tabletService.loadApplication(appName):
7                    self.logger.info("Successfully set application: %s" %appName)
8                    tabletService.showWebview()
9                    state = True
10               else:
11                   self.logger.warning("Got tablet service, but failed to set application: %
     s" %appName)
12           else:
13               self.logger.warning("Couldn't find tablet service, so can't set application:
     %s" %appName)
```

Listado 3.9: Selector de la ruta al paquete.

Sin embargo, existen otras maneras de utilizar la tablet, por ejemplo en este caso nos hemos enfocado en el uso de una carga de imágenes para aligerar el procesamiento de la tablet puesto que aunque podamos abrir la web de forma responsive, esto implica mayor carga de procesamiento y memoria, la cual viene muy limitada. En vista de estas circunstancias, la conclusión fue cargar imágenes con formato PNG.

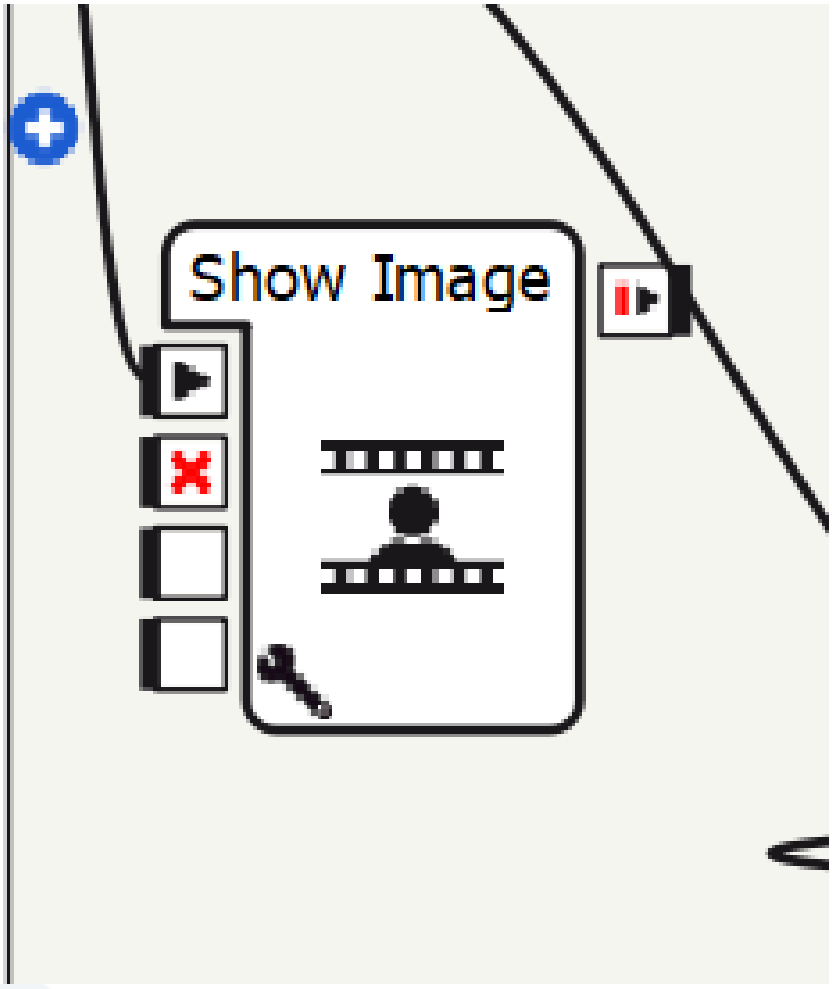


Figura 3.5: Carga de imagenes.

Para la carga de imágenes se ha usado una caja de Choreographe ya preparada para la subida de imágenes a la tablet. Esta caja viene con un código preensamblado que nos permite acceder a el paquete actual de la aplicación.

Este código define las rutas del paquete, de esta manera, se tiene un acceso a las carpetas de la aplicación, es decir, el HTML, el CSS o imágenes o cualquier archivo qué se requiera, como en este caso son imágenes, se accederá a la carpeta html donde existen imágenes guardadas.

En el siguiente código, usaremos la ruta de las imágenes donde introduciendo los parámetros en la caja de forma externa, se establecerá la imagen concreta que se quiere cargar en la tablet.

```
1
2  def onInput_onStart(self):
3        # We create TabletService here in order to avoid
4        # problems with connections and disconnections of the tablet during the life of the
     application
5        tabletService = self._getTabletService()
6        if tabletService:
7            try:
8                url = self.getParameter("ImageUrl")
```

```
                if url == '':
                    self.logger.error("URL of the image is empty")
                if not url.startswith('http'):
                    url = self._getAbsoluteUrl(url)
                tabletService.showImage(url)
            except Exception as err:
                self.logger.error("Error during ShowImage : %s " %err)
                self.onStopped()
        else:
            self.logger.warning("No ALTabletService, can't display the image.")
            self.onStopped()
```

Listado 3.10: Selector de la carpeta http y carga de una imagen.

Tras todo este proceso, la aplicación en el momento en el que ejecute la caja, cargará la tablet y mostrará la imagen con el nombre asociado al parámetro introducido.

La función de las imágenes es extender la explicación de cada zona que el robot explicará, de esta manera introduciremos una imagen de algún dispositivo destacable junto a algunas especificaciones técnicas.

Los archivos HTML creados están hechos con HTML 5 de manera responsive. Esto se ha hecho usando CSS en la cabecera de cada archivo. Sin embargo pese a que se podrían cargar directamente, se han hecho capturas de las distintas web y se han convertido en PNG.

4 Desarrollo del trabajo

4.1 Comienzo del trabajo

El desarrollo de este trabajo comenzó con el planteamiento del desarrollo del trabajo mostrado en el anteproyecto. En dicho documento, tratamos la idea de la función de anfitrión en el laboratorio Smarthome. Aquí se parte de la idea de qué trabajaríamos usando el simulador qué viene incluido en la herramienta Choreographe. El simulador qué viene incluido, fue de utilidad para las pruebas iniciales desde casa debido a la imposibilidad de ir a probar las aplicaciones en el robot ya qué el autor de este trabajo no vive en la provincia de Almería y debido a las restricciones [10], fue necesario adaptarse a estas circunstancias extraordinarias.

Se ha trabajado con una adaptación de la metodología Scrum como se comentó en el anteproyecto, sin embargo, los plazos cambiaron debido tanto a la incapacidad del simulador para recrear movimientos necesarios en el transcurso de la presentación como diversos fallos inherentes del sistema de NAOqi qué posteriormente se explicarán.

4.2 Puesta en marcha

En primer lugar, el robot vino en su caja junto al cargador y a un manual de usuario [3] que nos sirvió para la primera puesta a punto. Lo primero que hicimos fue sacarlo de la caja como se ve en la imagen 4.1 con precaución y comprobar su estabilidad.

Figura 4.1: Robot en su caja.

Tras sacar al robot de su caja el cual está con los frenos activados, se procede a retirar las dos llaves qué bloquean su uso movimiento. Manteniendo estas llaves tanto en el torso como en la cintura es la manera que tiene el robot de mantenerse en posición de seguridad cuando se necesita guardar. Hecho esto, podemos dejarlo en posición erguida como vemos en la siguiente imagen 4.2:

Figura 4.2: Desactivación del freno.

Con estas llaves 4.5, podemos poner el robot en marcha abriendo la goma plástica que tiene a la altura de la espalda como vemos en la siguiente imagen 4.3:

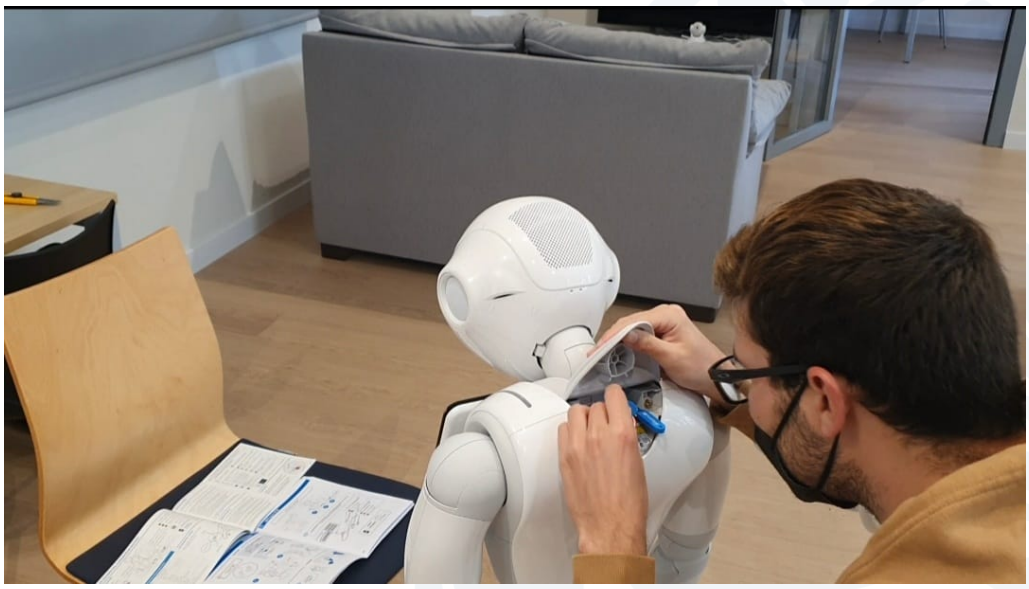


Figura 4.3: Desactivación de bloqueo.

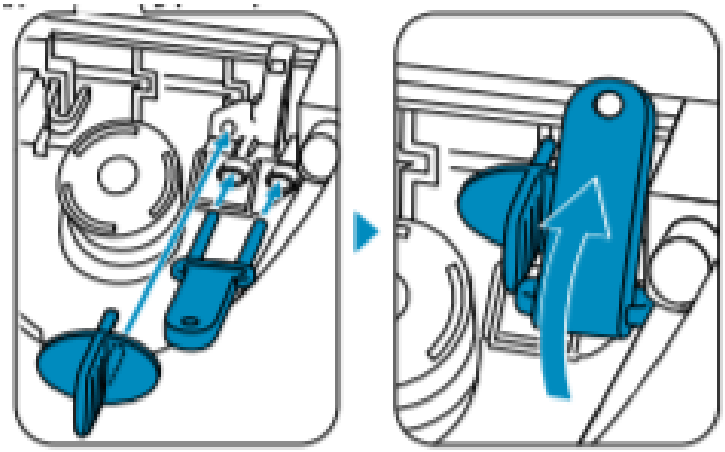


Figura 4.4: Esquema de conexión con las llaves. Fuente: [3]

Una vez puestas las llaves correctamente, debemos de girar el botón de parada de emergencia, el cual siempre se puede pulsar para parar el robot en caso de ser necesario.

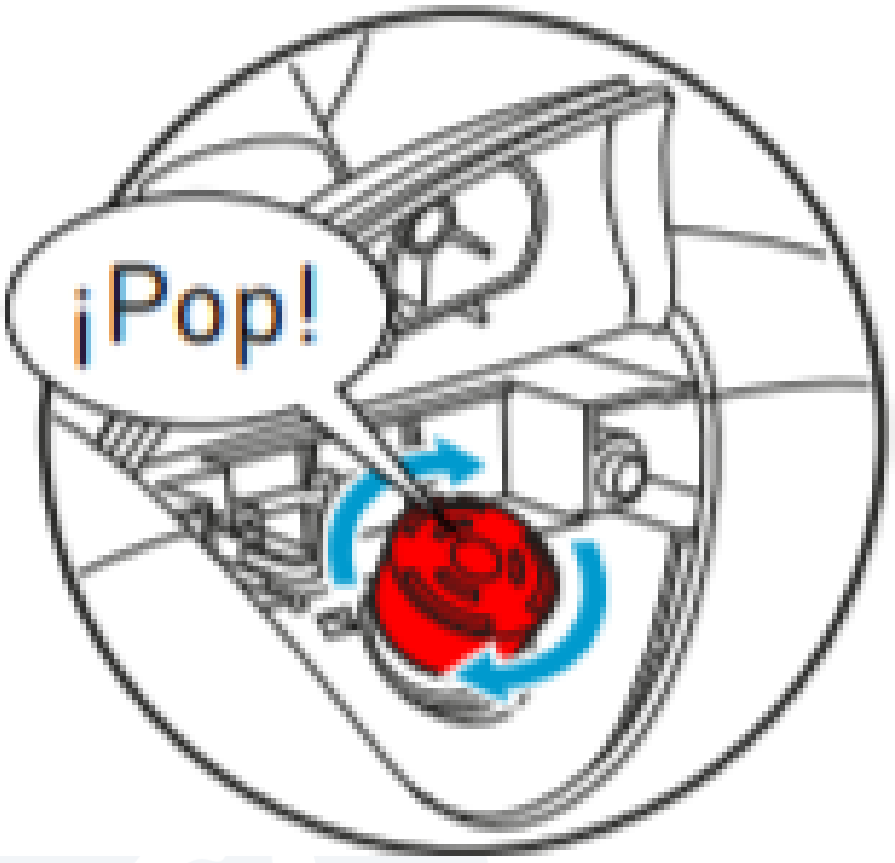


Figura 4.5: Freno de emergencia. Fuente: [3]

A la hora de cargar la batería, en caso de no usar una base como es nuestro caso, podemos usar el cargador, abrir la pestaña que tiene y conectar el cable como vemos en la siguiente imagen 4.6.

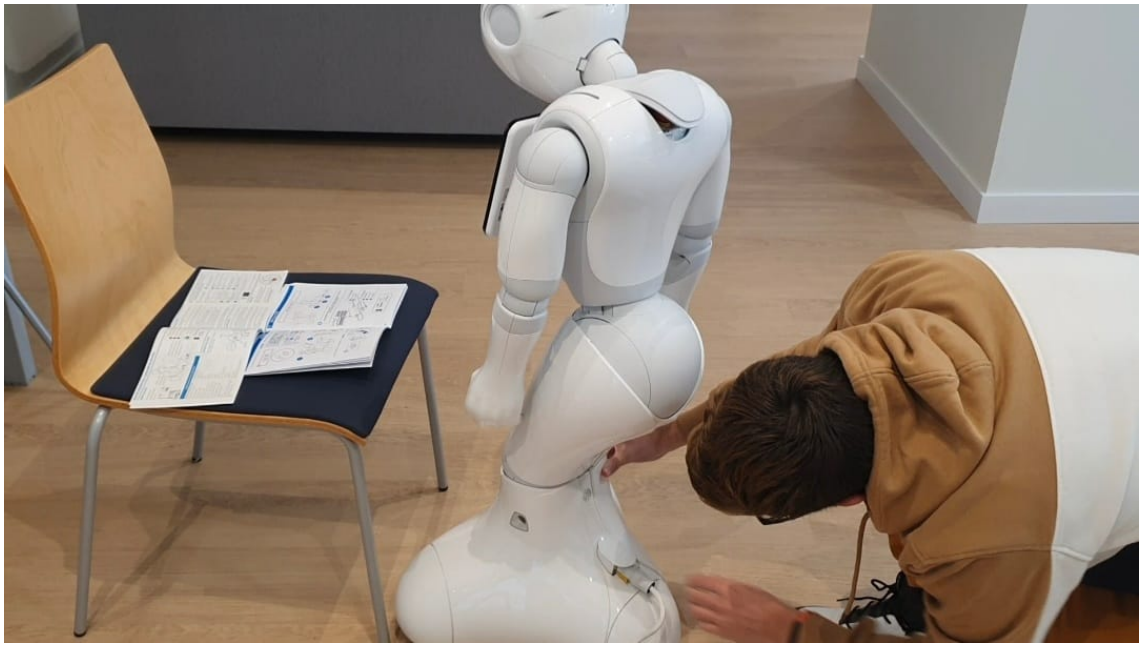


Figura 4.6: Robot cargando.

Como hemos podido comprobar las llaves 4.5 son un elemento importante en nuestro robot a la hora de arrancarlo, pero también nos sirve en el robot para abrir elementos como puede ser el interior de las ruedas o la cabeza para realizar cualquier tipo de mantenimiento.

Una vez con el robot preparado, podemos pulsar el botón que tiene en el pecho para encenderlo. Tardará alrededor de un minuto, tras esto estará en su modo de funcionamiento básico. Entonces podemos pulsar otra vez el botón de su pecho, entonces nos dirá cual es su IP, la cual sino se ha asignado antes, lo avisará. El robot puede funcionar sin tener una dirección IP, sin embargo, para acceder a el y modificar su comportamiento será necesario configurar una dirección IP. Para hacer esto, accederemos a través de un cable ethernet conectado directamente a su cabeza 4.7 desde nuestro ordenador, de esta manera, podremos ir a un navegador y conectarnos.

Figura 4.7: Cabeza abierta del robot.

Una vez dentro del panel web del robot, podemos verificar su versión, u actualizar las aplicaciones que vienen instaladas, sin embargo lo más importante es que podemos conectarnos vía WiFi a internet.

4.3 DESARROLLO DE LA APLICACIÓN

4.3.1 Desarrollo de la presentación desde casa

El trabajo se comenzó desde casa utilizando la herramienta Choreographe debido a qué esta incluía un simulador 4.8 permitiendo así ver algún tipo de salida.

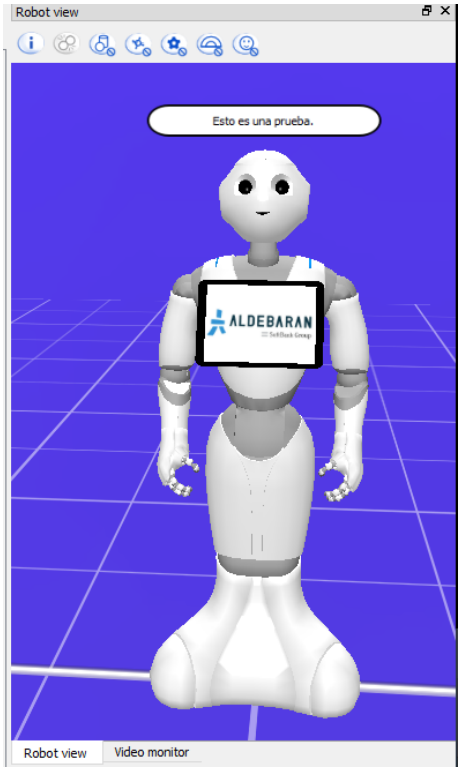


Figura 4.8: Imagen del simulador en funcionamiento.

Con el simulador funcionando se comienza el desarrollo de la presentación, esta se organiza primeramente en grupos de diagramas hechos por las 6 zonas que hemos considerado relevantes en el laboratorio: Entrada, sensores, cocina, lavabo, salón y dormitorio. La información para las distintas zonas se obtuvo gracias al director de este proyecto tomando referencias del vídeo **La computación nos invade** de la noche de los investigadores. Así pues se desarrolló un texto acorde a la presentación del laboratorio.

El texto se fue introduciendo en cada una de las cajas delimitadas por zonas incluyendo gesticulación al robot para poder dar naturalidad al hablar.

Desde aquí también probamos el servicio de **ALSpeechRecognition** [11] qué nos provee de la capacidad de reconocer palabras para luego establecer una respuesta. Este contenido se desarrollará fundamentalmente en el CTFG.

4.3.2 Desarrollo del contenido para la tablet

Tras el desarrollo de un contenido textual para el robot y las pruebas con el contenido de la sección Uso de el módulo de voz, el siguiente punto a tratar ha sido el tratado en Uso de la tablet. En este último, se planteo primero el desarrollo de una web, sin embargo, esta idea fue descartada debido a qué se tendría que desarrollar una web a partir de un *index* donde el usuario del robot pudiera interactuar con la tablet pero este no era el objetivo de la presentación ya que esto lo único que haría sería interrumpir la presentación. Por tanto, se desarrolló una página HTML con formato responsive para cada una de las secciones para posteriormente convertirlas en un PNG. Teniendo en cuenta esto, se procedió a cargar las imágenes correspondientes de cada zona una vez el robot comenzara a explicarlas.

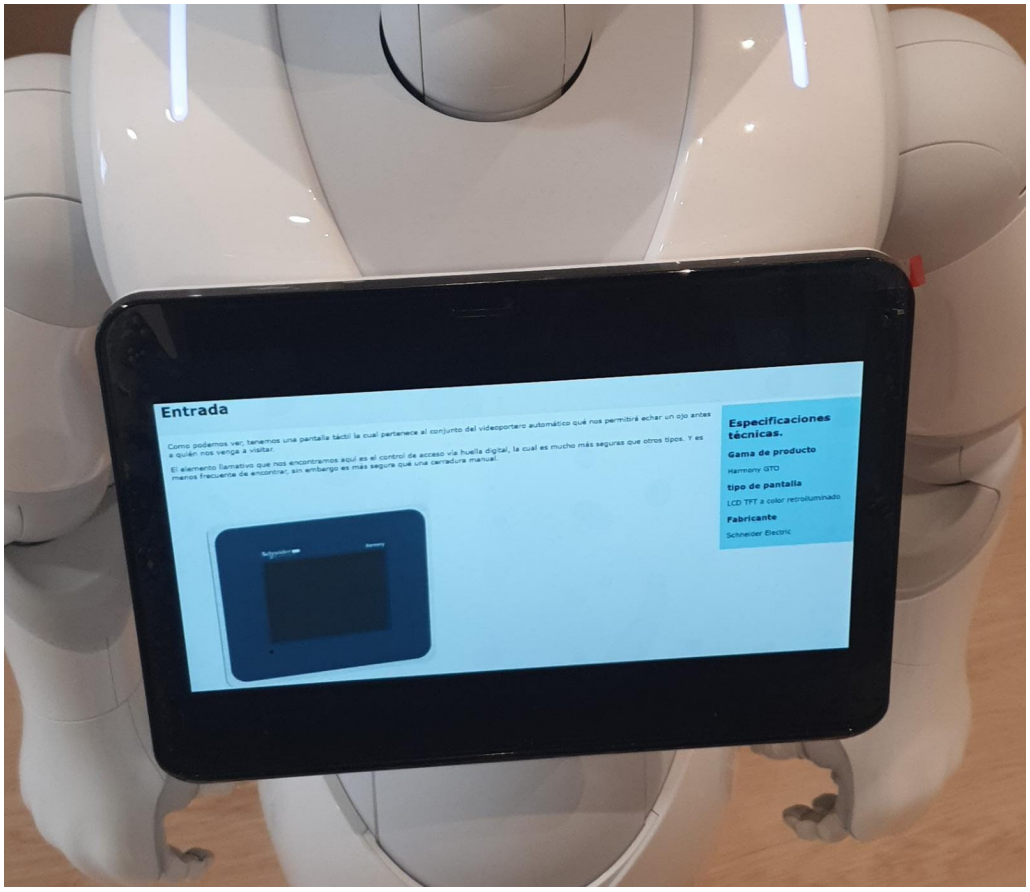


Figura 4.9: Tablet de robot Pepper mostrando parte de la presentación.

Cómo podemos observar en la figura 4.9, la tablet es un elemento vistoso que se usa para dar un énfasis a las distintas explicaciones que da. El uso de la tablet permite la posibilidad de poder leer la explicación qué el robot da a cualquier persona con problemas de capacidad auditiva, además aporta algunos datos técnicos extra para quién tenga curiosidad.

4.3.3 Desarrollo de la exploración

Tras tener una aplicación en la que el robot realizaba una explicación de las distintas zonas de la casa junto a otros detalles, se planteó la idea de que realizara movimientos. Para ello se planteó de distintas maneras, comenzando por poner movimientos fijos en una dirección concreta, pero trabajando esto in situ, fue rápidamente descartado al comprobar qué las rutas no las puede tomar siempre igual ya qué no tiene toda la precisión necesaria para esto

Recopilando información sobre Pepper, vimos como aparece en su página web un script que nos permitía crear un mapa, para esto usamos el script 3.5. En este caso, lo ejecutamos fuera de Choreographe, conectandonos vía SSH al robot. Tras copiar el script en el robot y ejecutarlo, este nos generó un mapa 3.2, sin embargo era ilegible a ojos humanos. Para solucionar este problema utilizamos un proyecto realizado por la empresa Aldebaran Robotics la cual nos ofrecía un código libre en su repositorio con el cual venían 3 proyectos para usar en Choreographe.

El primero era **explore** el cual funcionaba igual qué el script usado anteriormente. El proyecto más interesante del proyecto sin duda **places** como podemos ver en la figura 4.10, gracias a este proyecto, pudimos generar un mapa 3d en la tablet como el qué podemos ver en la figura 3.3, pero sin duda lo más útil fue poder cargar los distintos mapas y señalar lugares a través de la tablet, de este modo, se podría guardar las coordenadas de ese sitio y nombrarlas.

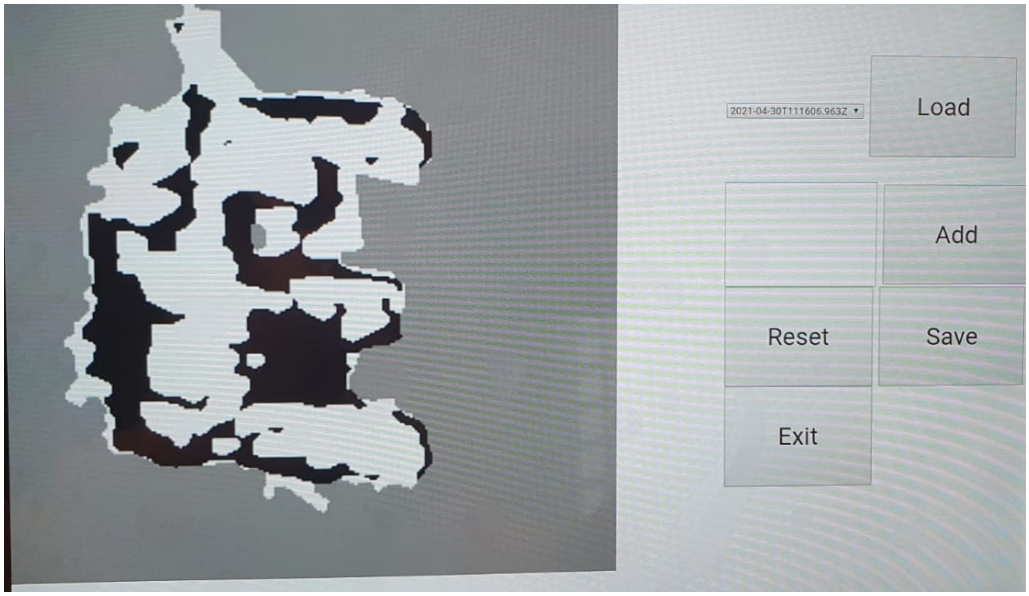


Figura 4.10: Aplicación places

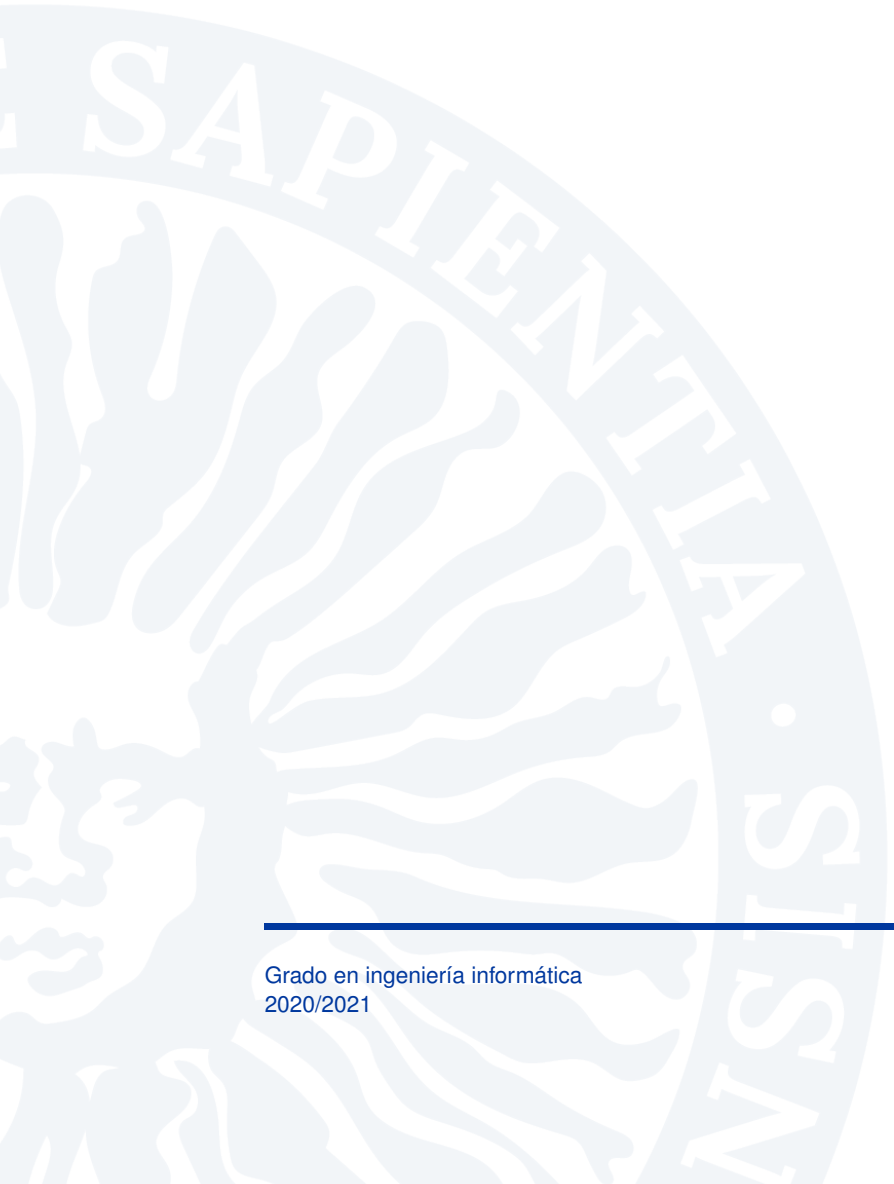
Como podemos ver, tenemos una opción para cargar los distintos mapas qué tenemos en el archivo donde se guardan los mapas. Una vez cargado el mapa, a ojo, se selecciona un punto de referencia relevante sobre el mapa y se escribe un nombre para añadir. Hecho esto, quedará marcado el sitio sobre el mapa. Una vez guardado esto, la aplicación nos devolverá las coordenadas asignadas a cada lugar en el siguiente formato:

```
places to save:'name':'2021-05-13T103104.394Z','places': 'Baño': [2.358142852783203, -3.382833480834961], 'Cocina': [-0.254968523979187, -3.8821539878845215], 'Salón': [4.555153846740723, -0.45348554849624634], 'Dormitorio': [5.8866753578186035, -3.2829692363739014], 'Puerta': [-0.8541533350944519, 0.07912316173315048]
```

4.3.4 Desarrollo de la navegación

Este proyecto lo podemos encontrar aquí: **Repositorio de Aldebaran**

5 Conclusiones



BIBLIOGRAFÍA

[1] Aliverobots, “Robot pepper,” Recuperado 1 Abril 2021 https://aliverobots.com/robot-pepper/, 2021.

[2] A. robotics, “Lenguajes soportados,” Recuperado 1 Abril 2021 http://doc.aldebaran.com/2-5/family/pepper_technical/languages_pep.html, 2021.

[3] S. robotics, “Guia de usuario,” Recuperado 1 Abril 2021 https://www.softbankrobotics.com/emea/sites/default/files/node/support/download/PEPPER_UserGuide_ES_2019%2003%2013_0.pdf, 2019.

[4] E. Garciéa Armada, *Robots / Elena Garciéa Armada.*, ser. Coleccioén Qué sabemos de? Madrid: Editorial CSIC Consejo Superior de Investigaciones Cientiéficas, 2015.

[5] A. K. Pandey and R. Gelin, “A mass-produced sociable humanoid robot: Pepper: The first machine of its kind,” *IEEE Robotics Automation Magazine*, vol. 25, no. 3, pp. 40–48, Sep. 2018.

[6] S. Femmam, “Nao robot,” in *Signals and Control Systems*, 1st ed. Hoboken, NJ, USA: Wiley, 2017, pp. 193–254.

[7] T. Chong, X. Tang, C. Leng, M. Yogeswaran, O. Ng, and Y. Chong, “Sensor technologies and simultaneous localization and mapping (slam),” *Procedia computer science*, vol. 76, pp. 174–179, 2015.

[8] A. robotics, “Motion,” Recuperado 2 Mayo 2021 http://doc.aldebaran.com/2-5/naoqi/motion/index.html, 2021.

[9] ——, “Almotion,” Recuperado 2 Mayo 2021 http://doc.aldebaran.com/2-5/naoqi/motion/almotion.html, 2021.

[10] J. de Andalucía, “Boletín oficial de la junta de andalucía,” Recuperado 18 Marzo 2021 https://www.juntadeandalucia.es/boja/2021/524/1, 2021.

[11] A. robotics, “Alspeechrecognition,” Recuperado 2 Mayo 2021 http://doc.aldebaran.com/2-5/naoqi/audio/alspeechrecognition.html, 2021.

A Tipos de referencias

Choreographe	Software utilizado.
Latex	Lenguaje utilizado para el desarrollo de la memoria.

Tabla A.1: Código software

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

En este archivo se incluirá tanto el resumen en castellano como su traducción al inglés. Los dos párrafos estarán ligeramente separados.

El propósito del trabajo en una o dos frases. El diseño y metodología utilizada, los resultados más significativos del trabajo realizado y un breve resumen de las conclusiones. Debe ser conciso y presentar los resultados obtenidos tras la ejecución del TFG.

Put here the english tranlation ... . Debe ser conciso y presentar los resultados obtenidos tras la ejecución del TFG. Los dos parrafos estarán ligeramente separados.

GRADO EN INGENIERÍA INFORMÁTICA

2020/2021