Introducción a la Robótica Inteligente

Trabajo Obligatorio 1

Grupo 1:

Javier López Iniesta Díaz del Campo
Jorge Quijorna Santos
Jorge Romeo Terciado

1 de Abril de 2020

Índice

1. Introducción

Tras afianzar nuestros conocimientos sobre arquitecturas basadas en el conocimiento, hemos basado nuestro proyecto en la implementación de un robot en el simulador *irsim* y cuyo comportamiento, corresponde con la arquitectura subsunción y con Motor Schemas. Con ello, se pretenderá resolver el problema planteado por el equipo, el cual se expondrá a continuación.

En nuestro caso, la tarea a resolver consiste en implementar el funcionamiento y comportamiento de un robot camarero de un restaurante. En él, un robot se encargará de atender a las distintas mesas. Estas mesas estarán representadas por las áreas negras. Además, se atenderá a unas mesas con prioridad ante otras si tienen encendidas en ellas una luz azul. La barra del bar será un área gris. Nuestro "camarero" tendrá que llevar los "pedidos" a la zona de las mesas y regresar a la barra directamente, sin atender antes a otros clientes. Antes de llegar a la zona del restaurante, se encontrará un laberinto, que simulará la entrada al mismo y que deberá atravesar para poder comenzar a atender. A la salida del laberinto, unas luces rojas reducirán la velocidad del robot. Por último, existirá una luz amarilla destinada para la carga de la batería del robot, y estará situada en la barra. Con ella conseguiremos que, la batería de la que dispone el robot se cargue. Todos estos detalles serán explicados con más detenimiento en los siguientes apartados.

Figura 1: Robot Camarero

2. Mapa

El mapa con el que se ha desarrollado este trabajo tiene las siguientes paredes como obstáculos, que simulan un laberinto, de entrada a un restaurante:

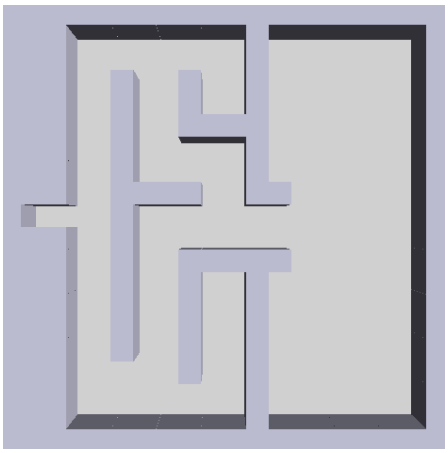


Figura 2: Paredes mapa

Además, hemos añadido los siguientes objetos en el archivo *o1.txt* que simulan un restaurante, para poder resolver el problema planteado, luces de distintos colores y varias zonas de distintos colores como parámetros:

- 1 luz amarilla: Su función principal es actuar como recarga de la batería del robot, además nos permitirá que el robot vuelva a la barra cada vez que atiende a un cliente (como se detallará más adelante).
- 5 luces azules: Se encontrarán justo encima de las mesas, y nos permitirá atender con mayor prioridad las mesas que tengan la luz encendida.
- 2 luces rojas: Nos permitirá reducir la velocidad de las ruedas, al salir del laberinto.
- 1 área gris: Se encargará de realizar la función similar a una barra del bar, donde el robot recogerá los pedidos para entregar a los clientes.
- 5 áreas negras: Serán las mesas de los clientes.

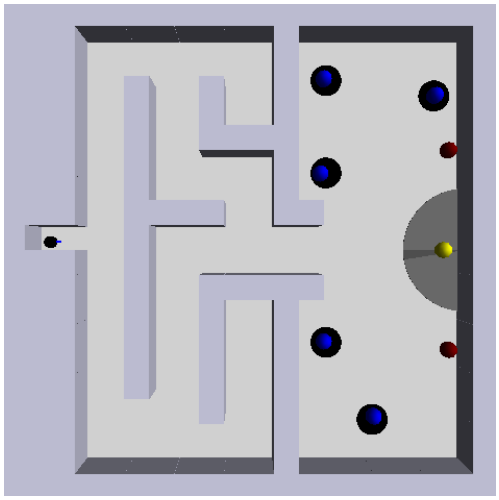


Figura 3: Objetos del experimento

3. Arquitectura

Las arquitecturas han sido divididas en cinco niveles de competencia distintos: "ObstacleAvoidance", "GoLoad", "Serve", "GoToBar" y "Navigate" (ver Figura 4 y 5):

- **ObstacleAvoidance**: Nivel de competencia que se encarga de hacer que el robot camarero evite los obstáculos. El nivel de competencia comprueba si hay algún obstáculo delante del robot, en caso de que si encuentre un obstáculo, calcula un vector en dirección opuesta al obstáculo que será enviado a los motores del robot.

- **GoLoad**: Este nivel de competencia se encargará de comprobar el estado de la batería del robot. Si el nivel de batería es inferior a un cierto umbral, se orienta hacia la fuente de luz amarilla.

- **Serve**: El nivel de competencia de Serve comprueba el estado del sensor de la pinza (bandeja del camarero). En caso de que no se encuentre activo, ya sea porque no tiene ningún objeto o si la señal está siendo inhibida, no realiza ninguna acción. Si se encuentra activo realiza una función parecida a *Load Battery* orientándose hacia una fuente de luz azul.

- **GoToBar**: Nivel de competencia que se encarga de ir hacia la barra del bar. Comprueba el estado de la pinza, si la señal esta siendo inhibida y si se encuentra en la parte del restaurante (luz roja). Si se cumple está condición, se orienta hacia la fuente de luz amarilla.

- **Navigate**: Este nivel de competencia pone una velocidad *SPEED* (1000 en nuestro caso) al robot en ambos motores, si detecta un valor total en todos sus sensores de luz roja mayor a un cierto umbral, reduce la velocidad a *SPEED - 300* (700 en nuestro caso). Al ser la velocidad igual en ambos motores, el robot se moverá en línea recta, mientras este nivel de competencia esté activado.

Todos estos niveles de competencia, se analizarán y se explicarán posteriormente en la parte de implementación, en los archivos *iri1controller.cpp* para el caso de cada arquitectura.

A continuación, se puede observar un esquema asociado a la arquitectura de subsunción y a la de Motor Schemas:

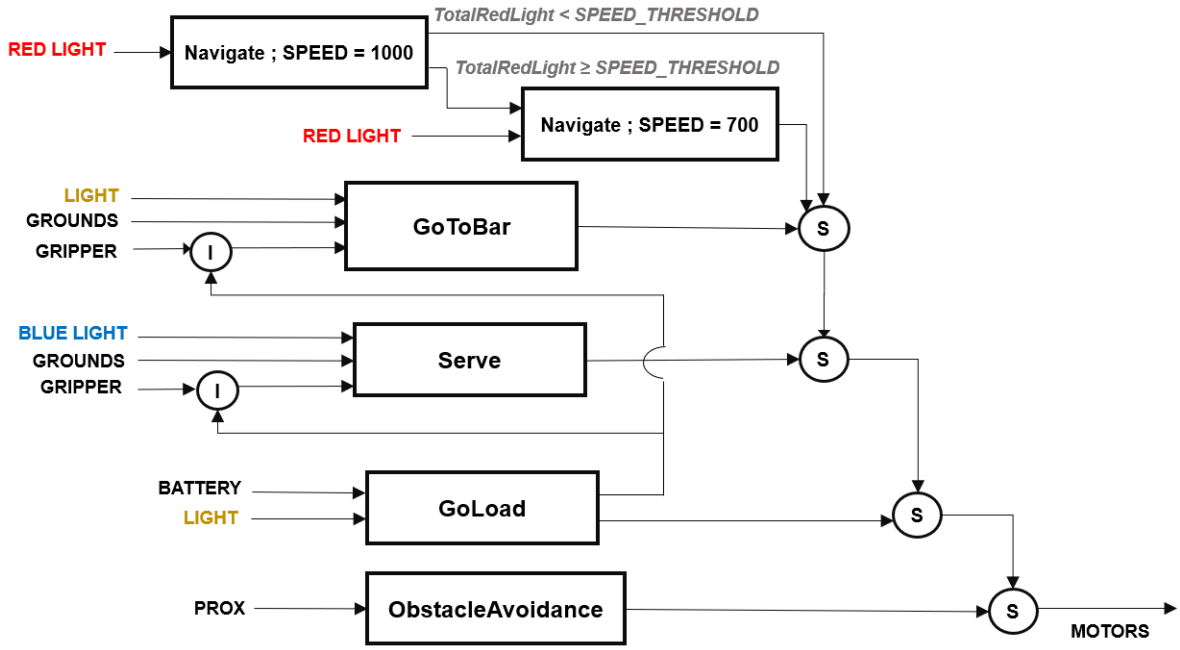


Figura 4: Esquema arquitectura subsunción

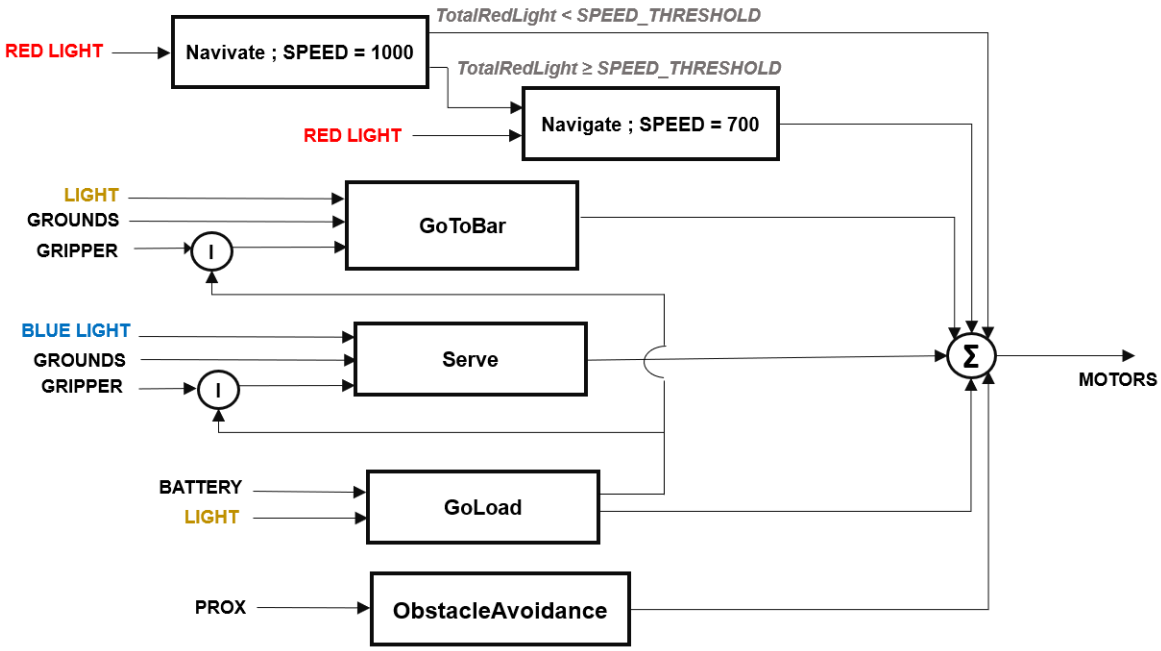


Figura 5: Esquema Motor Schemas

4. Implementación

Para la implementación del proyecto realizado, se han realizado las siguientes variaciones en los ficheros que se detallan a continuación:

4.1. iri1exp.cpp

Este experimento utiliza los motores, sensores de luz, sensor de proximidad, sensor de suelo de memoria y sensor de batería. Este archivo se encarga de crear a partir del fichero de parámetros, la arena, el robot y todos los distintos objetos.

En el experimento, se crea una arena de $3x3\,m^2$ y se introduce el robot en la posición (-1.25, 0.05) para que tenga que cruzar el laberinto antes de llegar al restaurante. Dicho mapa es el mismo para ambas arquitecturas, aunque para el caso de Motor Schemas se ha hecho una pequeña modificación para facilitar la salida del laberinto en poco tiempo. Se ha implementado mediante un array de porcentajes y almohadillas como se muestra a continuación:

```
"%%%%%%%%%%%%%%%%%%%%"
"%%%########%#######%"
"%%%########%#######%"
"%%%#####%##%#######%"
"%%%##%##%##%#######%"
"%%%##%##%##%#######%"
"%%%##%##%##%#######%"
"%%%##%##%##%#######%"
"%%%##%##%%%%######%"
"%%%##%#############%"
"%####%#############%"
"%%%##%%%%##%######%"
"%%%##%#####%#######%"
"%%%##%#####%#######%"
"%%%##%##%%%#######%"
"%%%##%##%##%#######%"
"%%%##%##%##%#######%"
"%%%########%#######%"
"%%%########%#######%"
"%%%%%%%%%%%%%%%%%%%%";
```

Figura 6: Arena experimento para arquitectura subsunción

Dichos experimentos, tanto el correspondiente a la arquitectura de subsunción, como el de Motor Schemas, se ejecutan desde el simulador con la instrucción:

./irsim -E 30 -p paramFiles/o1.txt

4.2. iri1controller.cpp

Para la realización de este fichero nos hemos basado en el ejemplo proporcionado en **Robolabo** (`http://robolabo.etsit.upm.es/subjects.php?subj=irin&tab=tab3&lang=es`) de *subsumption-garbagecontroller.cpp*, para el caso de la arquitectura de subsunción. Así como el fichero *motorSchemas2controller.cpp* par el caso de Motor Schemas.

Además, hemos definido tres diferentes constantes, que actuarán como umbrales en el experimento:

- PROXIMITY_THRESHOLD = 0.6, dicho valor se lo hemos asignado ya que le permitía al robot salir del laberinto y evitar los obstáculos de una forma adecuada.
- BATTERY_THRESHOLD = 0.15 para el caso de subsunción. Sin embargo, en Motor Schemas hemos variado este umbral, siendo igual a 0.45 para comprobar el correcto funcionamiento de la batería, y que esta, se descargue al menos una vez para observar el comportamiento de *GoLoad*.
- SPEED_THRESHOLD = 0.6, el robot reduce la velocidad al salir del laberinto, cuando el total de los sensores de luz roja superan este umbral.

Definimos en la memoria M_{left} y M_{right}, como las velocidades del motor del lado izquierdo y derecho respectivamente. La salida del módulo será el vector $\vec{M} = (M_{left}, M_{right})$. En adicción, también se ha definido una constante para establecer el valor inicial de la velocidad de cada motor, es SPEED = 1000.

4.2.1. Arquitectura subsunción

Para el caso de la arquitectura de subsunción, los diferentes niveles de competencia implementados en el proyecto, no se ejecutan en paralelo. Por ello, es necesario crear un coordinador que se encargue de inhibir/suprimir las tareas.

El coordinador basa su funcionamiento en una tabla donde están recogidas los valores de las salidas de cada uno de los distintos niveles de competencia, así como un flag de activación. Cada nivel de competencia, se ejecuta y pone su velocidad al motor de cada rueda en la fila correspondiente. Si el nivel de competencia va a suprimir a otros niveles de competencia, pone su flag a TRUE.

Una vez que se ejecutan todos los niveles de competencia, el coordinador comprueba en la tabla desde el nivel más bajo, al nivel más alto. Cuando encuentra un flag activado en determinado nivel de competencia, le pasa a los actuadores la velocidad de dicho nivel.

4.2.2. Motor Schemas

En cambio en Motor Schemas, los distintos niveles de competencia, se pueden ejecutar en paralelo. Por lo que el funcionamiento del coordinador en este caso, comprueba si está activo algún nivel de competencia, en dicho caso se calcula la velocidad de ambos motores:

En primer lugar, definimos en la memoria $m\_fActivationTable[0]$, como SL y $m\_fActivationTable[0]$ como SR, y se calcula el angulo, φ, mediante la siguiente ecuación:

$$\varphi = arctg\left(\frac{\sum_{i=0}^{nBehavior} SR_i * sen(SL_i)}{\sum_{i=0}^{nBehavior} SR_i * cos(SL_i)}\right) \tag{1}$$

Posteriormente, se normaliza dicho ángulo entre $[-\pi, \pi]$. Y se establece la velocidad en los motores del robot, mediante la siguiente ecuación:

$$\vec{M} = \left(SPEED * cos\left(\frac{\varphi}{2}\right) - \frac{SPEED}{\pi} * \varphi,\ SPEED * cos\left(\frac{\varphi}{2}\right) + \frac{SPEED}{\pi} * \varphi\right) \tag{2}$$

A continuación se explica, como se ha desarrollado cada nivel de competencia implementado:

4.2.3. ObstacleAvoidance

Este nivel de competencia se encarga de que el robot no colisione con las paredes del simulador. En él se leen los valores de los sensores de infrarrojos ($IR_{Si} \in [0,1], i \in [0,7]$) con una posición angular θ_{IRi} para cada uno de los sensores del robot. El módulo se encarga de calcular la orientación a la que se encuentra el obstáculo más cercano (φ), mediante el valor de cada uno de los sensores y sus posiciones angulares:

$$\varphi = \pi - arctg\left(\frac{\sum_{i=0}^{7} IR_i * sen(\theta_{IR_i})}{\sum_{i=0}^{7} IR_i * cos(\theta_{IR_i})}\right) \tag{3}$$

Dicho angulo se normaliza entre $[-\pi, \pi]$. Por otra parte, se calcula el valor máximo medido por los sensores de proximidad del robot. Dicho valor se compara con un umbral llamado *PROXIMITY_THRESHOLD*, si lo supera, cambia el color del robot a verde y establece la velocidad en los motores del robot, dada por la siguiente ecuación:

$$\vec{M} = \left(SPEED * cos\left(\frac{\varphi}{2}\right) - \frac{SPEED}{\pi} * \varphi,\ SPEED * cos\left(\frac{\varphi}{2}\right) + \frac{SPEED}{\pi} * \varphi\right) \tag{4}$$

4.2.4. GoLoad

Este nivel de competencia se encarga de proporcionar al robot una batería, que se descargue y vaya hacia la luz amarilla a cargarse. En dicho método se leen los 8 sensores de luz amarilla del robot ($S_{yellow_i} \in [0,1], i \in [0,7]$) con una posición angular θ_{IRi}, respecto al sentido del movimiento del robot. Una luz solamente puede activar 2 sensores de luz, estos son los más cercanos a dicha luz. Además el robot cuenta con una batería($B_{Y1} \in [0,1]$), donde 0 indica que esta totalmente descargada la batería y 1 que está esta cargada al 100 %.

Está función se encarga de leer el nivel de la batería. Si dicho nivel es superior a un umbral, conocido como *BATTERY_THRESHOLD*, no realiza ninguna acción ($B_{Y1} \geq BATTERY\_THRESHOLD$). En cambio, si el nivel de la batería es inferior al umbral ($B_{Y1} < BATTERY\_THRESHOLD$) no dispone de la suficiente batería para seguir realizando el resto de acciones, y se realizan las siguientes acciones:

1. Se cambia el color del robot a rojo.
2. Se activa el inhibidor(activo a nivel bajo) conocido como *fBattToServeInhibitor*.
3. Se comprueba la orientación del robot en el sentido de la luz. Si el robot ya se encuentra orientado($S_{yellow_1} \neq 0$ y $S_{yellow_7} \neq 0$), no se realiza ninguna acción. En cambio, si este no está orientado trata de orientarse respecto a la luz. Esto se consigue guardando en dos variables los datos de los sensores de la izquierda y la derecha respectivamente:

$$S_{left} = \sum_{i=0}^{3} S_{yellow_i} \tag{5}$$

$$S_{right} = \sum_{i=4}^{7} S_{yellow_i} \tag{6}$$

Además, se varía la velocidad de los motores del robot de la siguiente manera:

$$\vec{M} = \begin{cases} (-SPEED, SPEED) & si\ S_{left} > S_{right} \\ (SPEED, -SPEED) & si\ S_{left} \leq S_{right} \end{cases} \tag{7}$$

4.2.5. Serve

Este nivel, se encarga de transportar la comida (objetos virtuales) desde la barra (zona gris) hasta las mesas (zonas negras). El robot dispone de un sensor pinza (groundMemory[0]), llamado $G_{S1} \in 0, 1$. Dicha pinza se activa (1) cuando el robot coge un pedido de la barra, y permanece desactivado (0) cuando no tienen ningún pedido. En primer lugar, se comprueba si serve está activo o no, ya sea porque la pinza no tiene ningún pedido o porque la señal está siendo inhibida. En caso de que no se encuentre activo, ya sea porque no tiene ningún objeto o si la señal está siendo inhibida, no realiza ninguna acción. Si se encuentra activo realiza una función parecida a "Load Battery" orientándose hacia una fuente de luz azul.

Se cambia el color del robot a azul, además se activa (1) la variable *TurnOffBlueLight*, que nos permitirá apagar una luz azul si se entrega un pedido (implementado en GoToBar). También, se leen los 8 sensores de luz azul del robot ($S_{bluei} \in [0, 1], i \in [0, 7]$) con una posición angular θ_{IRi}, respecto al sentido del movimiento del robot. Cada mesa cuenta con una luz azul, que le permitirá gozar de la prioridad a la hora de recibir los pedidos por parte del camarero. Las mesas más cercanas a la barra, serán atendidas antes, que las que se encuentren más alejadas.

Además, se comprueba la orientación del robot en el sentido de la luz. Si el robot ya se encuentra orientado($S_{blue1} \neq 0$ y $S_{blue7} \neq 0$), no se realiza ninguna acción. En cambio, si este no está orientado trata de orientarse respecto a la luz azul. Esto se consigue guardando en dos variables los datos de los sensores de la izquierda y la derecha respectivamente:

$$S_{left} = \sum_{i=0}^{3} S_{blue_i} \tag{8}$$

$$S_{right} = \sum_{i=4}^{7} S_{blue_i} \tag{9}$$

Además, se varía la velocidad de los motores del robot de la siguiente manera:

$$\vec{M} = \begin{cases} (-SPEED, SPEED) & si\ S_{left} > S_{right} \\ (SPEED, -SPEED) & si\ S_{left} \leq S_{right} \end{cases} \tag{10}$$

4.2.6. GoToBar

Este nivel de competencia se encargará de regresar a la barra del bar, inmediatamente después de entregar un pedido a un cliente o al llegar a la zona del restaurante. Realiza las siguientes acciones:

En primer lugar, se leen los sensores de luz roja del robot, y se almacenan en una variable:

$$fTotalRedLight = \sum_{i=0}^{7} S_{red_i} \tag{11}$$

A continuación, si se cumple la condición de que: la pinza no tenga ningún objeto, la señal no esté siendo inhibida y que se encuentre en la zona del restaurante ($fTotalRedLight > SPEED\_THRESHOLD$), se realizan las siguientes acciones:

1. Si la variable *TurnOffBlueLight* está puesta a 1, apaga la luz azul más cercana y pone *TurnOffBlueLight* a 0, con el fin de que solo pueda apagar la luz azul de la mesa que ha sido atendida.

2. Se comprueba la orientación del robot en el sentido de la luz amarilla. Si el robot ya se encuentra orientado($S_{yellow_1} \neq 0$ y $S_{yellow_7} \neq 0$), no se realiza ninguna acción. En cambio, si este no está orientado trata de orientarse respecto a la luz. Esto se consigue guardando en dos variables los datos de los sensores de la izquierda y la derecha respectivamente:

$$S_{left} = \sum_{i=0}^{3} S_{yellow_i} \tag{12}$$

$$S_{right} = \sum_{i=4}^{7} S_{yellow_i} \tag{13}$$

Además, se varía la velocidad de los motores del robot de la siguiente manera:

$$\vec{M} = \begin{cases} (-SPEED, SPEED) & si\ S_{left} > S_{right} \\ (SPEED, -SPEED) & si\ S_{left} \leq S_{right} \end{cases} \tag{14}$$

4.2.7. Navigate

Este nivel de competencia establece una velocidad constante a ambos motores del robot, permitiendo que se mueva en línea recta. Ha sido implementado en el método *Navigate*. Además, en este nivel de competencia se leen los sensores de luz roja del robot, y se almacenan en una variable:

$$fTotalRedLight = \sum_{i=0}^{7} S_{red_i} \tag{15}$$

La función que realiza este nivel de competencia se puede describir mediante la siguiente ecuación:

$$\vec{M} = \begin{cases} (SPEED, SPEED) & si\ fTotalRedLight < SPEED\_THRESHOLD \\ (SPEED - 300, SPEED - 300) & si\ fTotalRedLight \geq SPEED\_THRESHOLD \end{cases} \tag{16}$$

Como las 2 luces rojas, se encuentran a la derecha del mapa, y tienen un rango del sensor limitado a 1.5 metros, en la zona del laberinto no se detectará apenas luz. Por tanto, la velocidad de los motores del robot será:

- **Velocidad 1000**: Los motores del robot se mueven a dicha velocidad, mientras se encuentra dentro del laberinto.
- **Velocidad 700**: Al salir, del laberinto, el robot camarero reduce la velocidad de los motores en 300. Esa velocidad se mantiene mientras el robot esté en la zona del restaurante (mesas y barra).

4.3. iri1controller.h

En este fichero se han declarado, los sensores que se han utilizado. Además, también se han definido los nuevos niveles de competencia (*Serve* y *GoToBar*) para que puedan ser utilizados en el fichero ***.cpp***, así como las variables globales que se han utilizado en este proyecto. Cabe destacar, *fBattToServeInhibitor* o *TurnOffBlueLight*, ambas de tipo double.

4.4. Objetos

4.4.1. blueLightObject.cpp

Las luces azules estaban inicialmente pre diseñadas para que estuvieran permanentemente encendidas. Sin embargo, en nuestro proyecto, cuando el robot camarero atiende a una mesa, apaga su luz. Cuando el robot apaga todas las luces, existe una necesidad de volver a encenderlas. Por ello se ha definido en el método *GetTiming* un temporizador para que cada 1000 timesteps encienda todas las luces azules, independientemente si están encendidas o apagadas.

4.4.2. redLightObject.cpp

En cambio, las luces rojas inicialmente permanecían realizando un parpadeo, debido a esto, el robot no era capaz de disminuir la velocidad una vez que saliera del laberinto. Por ello, lo hemos solucionado comentando la parte del código del método *GetTiming* para que ahora la luz roja permanezca encendida permanentemente.

4.5. Archivo de parámetros

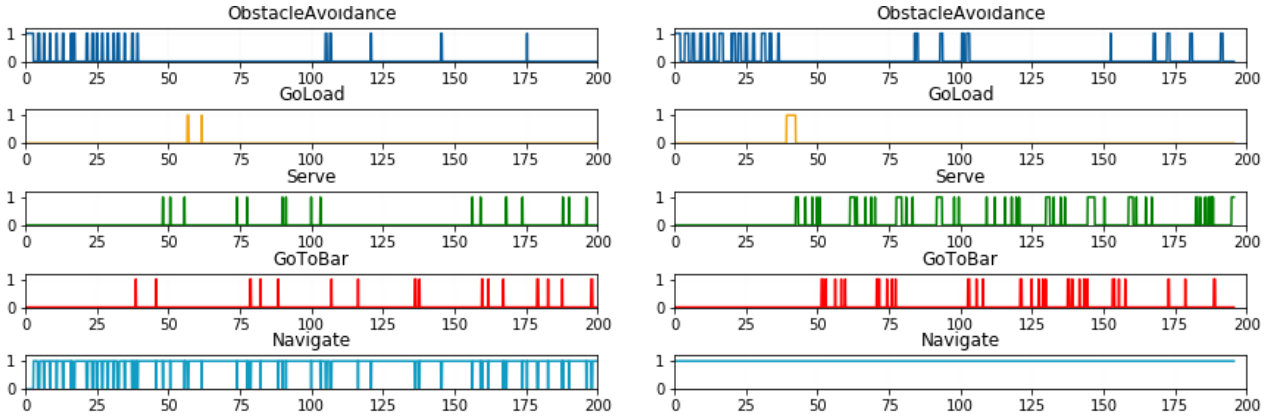
Como ya se ha comentado anteriormente, se han definido todos los objetos del experimento en el archivo *o1.txt*, así como la posición inicial del robot, para que tenga que atravesar el laberinto para llegar al restaurante.Además, se han establecido los distintos niveles de carga y descarga de las luces.

5. Gráficas

Para realizar las gráficas, utilizamos los datos generados a partir de todos los sensores del robot almacenados en los ficheros de la carpeta *outputFiles*, así como haciendo uso de *Python* y sus distintas librerías (pandas, matplotlib y seaborn). Analizando dichos datos, a partir de los distintos sensores del robot, hemos realizado las siguientes gráficas, que nos permiten analizar las distintas arquitecturas de forma más específica.

A continuación, se presentan las gráficas realizadas:

En el siguiente gráfico, se observa la activación de los diferentes niveles de competencia, tanto en el caso de la arquitectura de subsunción, como la de Motor Schemas:

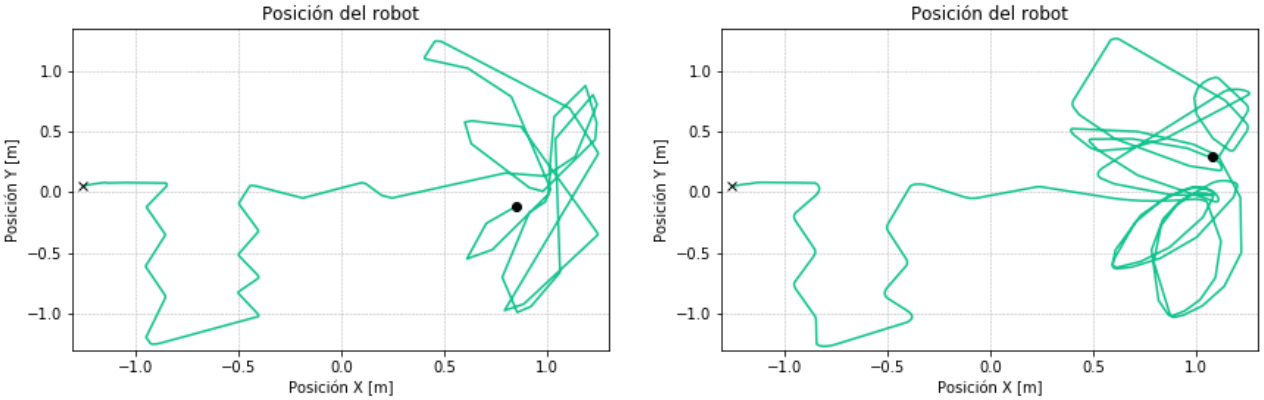


(a) Arquitectura subsunción (b) Motor Schemas

Figura 7: Activación niveles de competencia

Se puede observar como, en la arquitectura de subsunción, solo hay activo un único nivel de competencia. Mientras, en Motor Schemas pueden estar activos a la vez varios niveles de competencia.

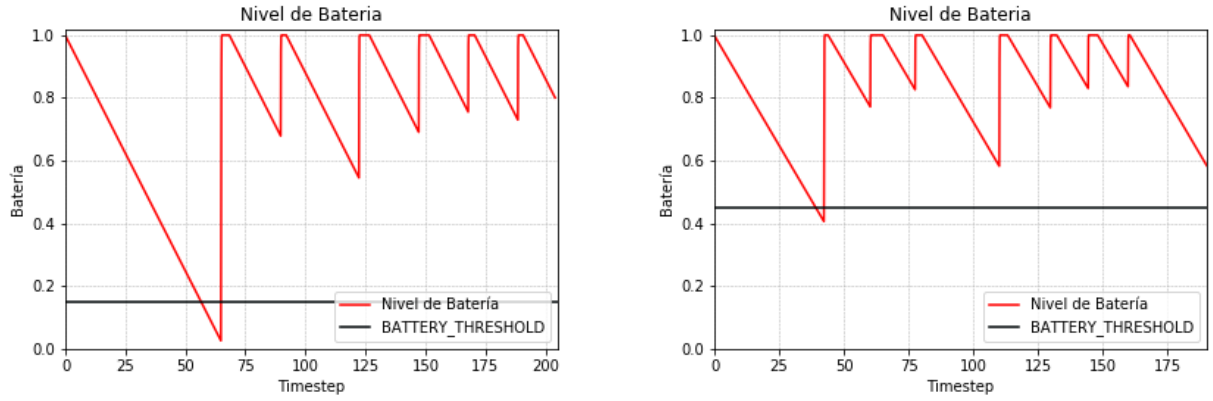
Además también se ha representado la trayectoria del robot. Se puede observar, el punto de inicio marcado por una "X", así como el punto final marcado por un círculo.



(a) Arquitectura subsunción (b) Motor Schemas

Figura 8: Trayectoria robot

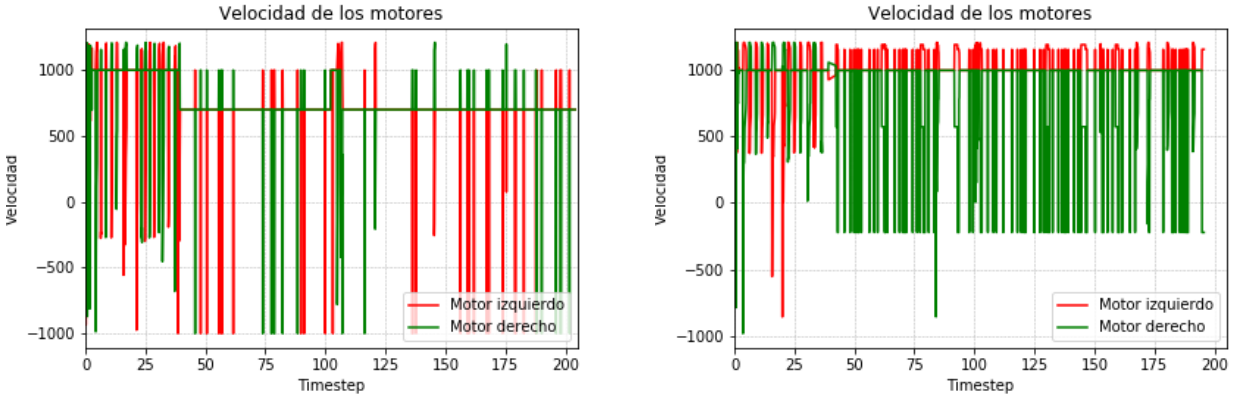
La siguiente gráfica, muestra la variación del nivel de la batería en función del tiempo (*Timesteps*), observando el tiempo de carga y descarga:



(a) Arquitectura subsunción (b) Motor Schemas

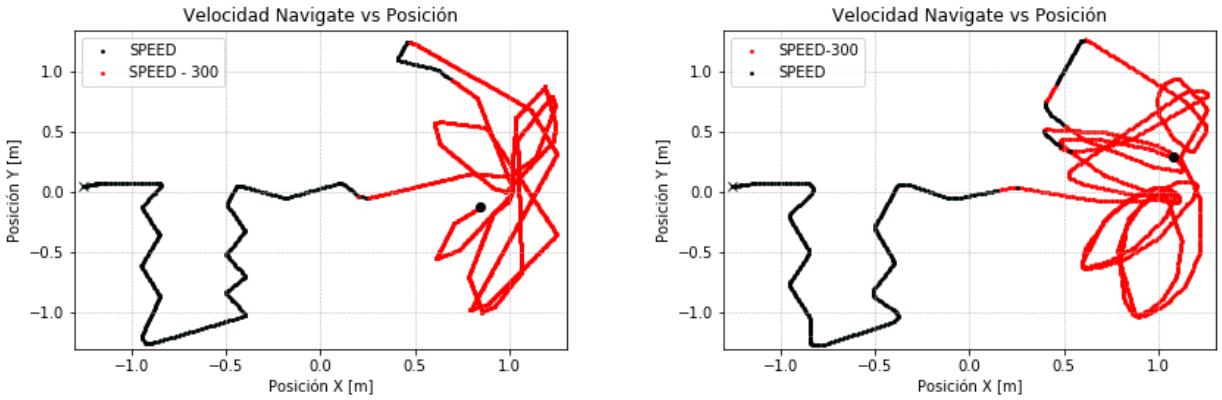
Figura 9: Nivel de batería

A continuación, se muestra la velocidad del motor de cada una de las ruedas del robot (Figura 9). Además, también hemos representado el valor de la velocidad en el nivel de competencia *Navigate*, si el total de los sensores de luz roja, supera un umbral (rojo) o no (negro):



(a) Arquitectura subsunción (b) Motor Schemas

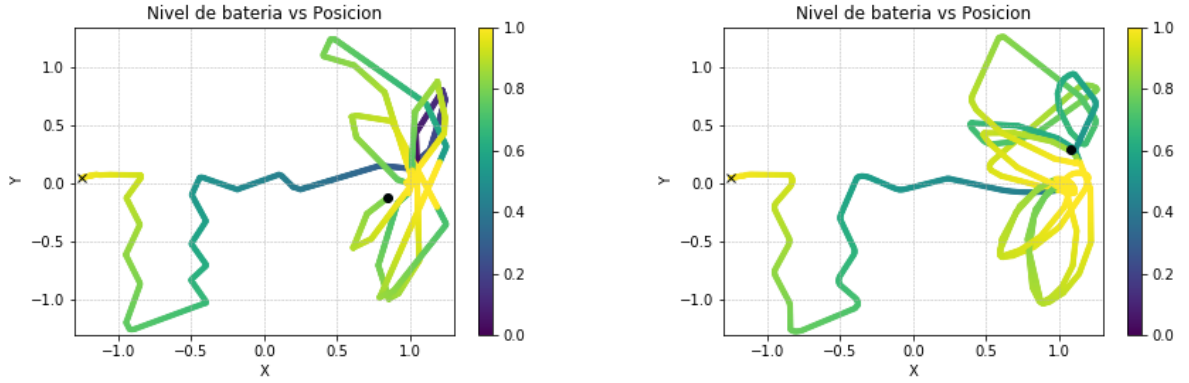
Figura 10: Velocidad ruedas



(a) Arquitectura subsunción (b) Motor Schemas

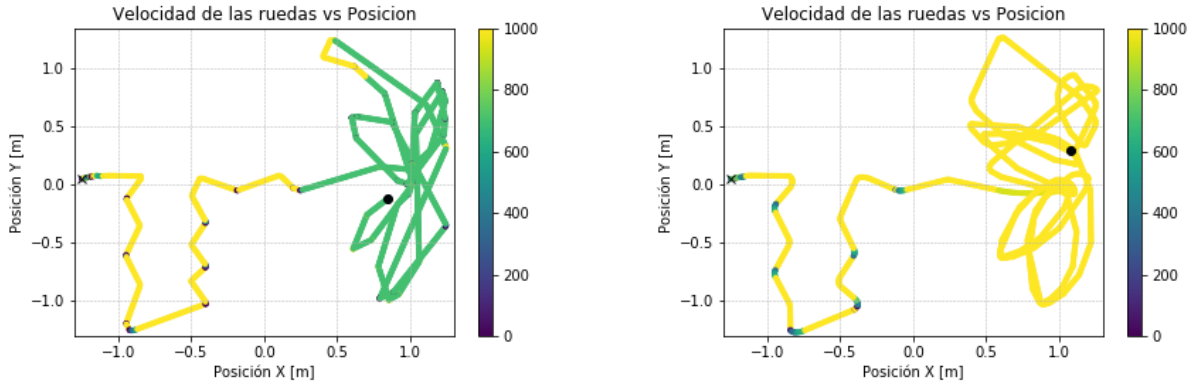
Figura 11: Velocidad Navigate

Los siguientes gráficos, están relacionados con la posición del robot durante la simulación. Hemos representado el nivel de batería (Figura 12) y velocidad de las ruedas (Figura 13), en función de la posición (coordenadas X e Y):



(a) Arquitectura subsunción (b) Motor Schemas

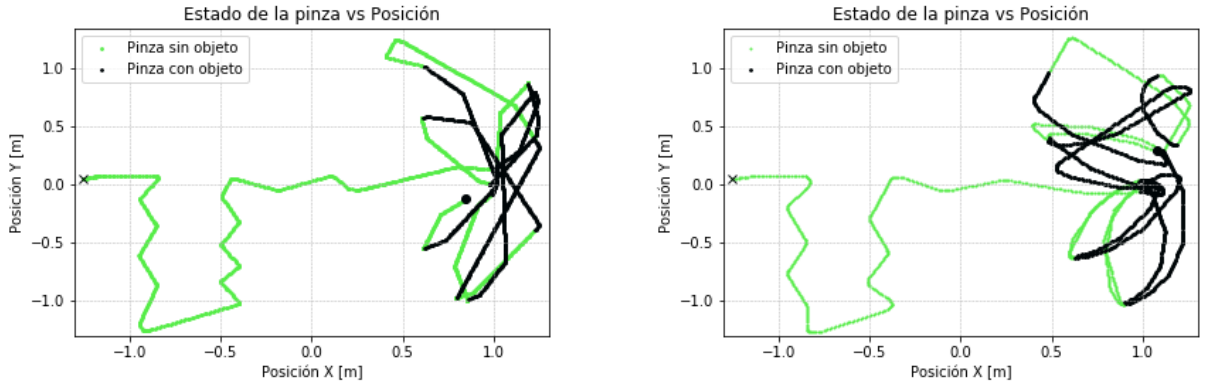
Figura 12: Nivel de Batería vs Posición



(a) Arquitectura subsunción (b) Motor Schemas

Figura 13: Velocidad de las ruedas vs Posición

Asimismo, se ha representado el valor de la pinza del robot (0 verde y 1 negro), para observar que cuando se desplaza hacia una mesa, lleva un objeto y cuando se dirige hacia la barra, no lleva nada:



(a) Arquitectura subsunción (b) Motor Schemas

Figura 14: Velocidad de las ruedas vs Posición

6. Comparación

Para realizar la comparación entre la arquitectura subsunción y la arquitectura Motor Schemas nos vamos a fijar en dos aspectos fundamentales: arquitectura y funcionamiento.

Si observamos la Figura 4 y la Figura 5 encontramos la primera diferencia entre ambas arquitecturas, mientras que en la arquitectura subsunción cada nivel de comportamiento actúa sobre los motores de manera independiente, en la arquitectura Motor Schemas se suman todos los comportamientos antes de actuar sobre los motores. Cabe destacar, que mientras en la arquitectura subsunción únicamente se ejecuta un nivel de competencia, en Motor Schemas pueden llegar a ejecutarse varios niveles en paralelo. Estas diferencias entre las arquitecturas se programan en el coordinador de cada uno de los controladores.

En consecuencia, al ejecutar las dos arquitecturas observamos diferencias significativas en su funcionamiento. Como se puede observar en la Figura 8, al llegar a un obstáculo, la arquitectura de subsunción rota sobre si misma para después avanzar mientras que la arquitectura Motor Schemas realiza un giro más suave tanto en la zona de laberinto como en la zona de restaurante, esto hace que hayamos tenido que introducir una pequeña variación entre el laberinto que resuelve la arquitectura subsunción y el laberinto que resuelve la arquitectura Motor Schemas para que ninguna de las arquitecturas tenga problemas en resolver el laberinto. Por otro lado, en la zona de restaurante, la arquitectura subsunción va más directa a las mesas mientras que Motor Schemas realiza una trayectoria haciendo giros, de tal forma que el tiempo que tardan las dos arquitecturas en servir las 5 mesas es distinto, como era de esperar.

Por último, hay que destacar que no hay una arquitectura mejor que otra, ambas tienen la misma funcionalidad, pero su forma de proceder es diferente. Por ello, si tuviéramos que elegir una arquitectura, tendríamos que fijarnos en qué aspectos consideramos importantes en cada caso. En el caso particular de nuestro experimento, podríamos diferenciar entre la zona de laberinto y la zona de restaurante. Quizá en la zona de laberinto sería más apropiado utilizar la arquitectura Motor Schemas puesto que hace los giros más suaves que la arquitectura de subsunción, pero en la zona de restaurante sería más apropiado utilizar una arquitectura de subsunción, ya que, como se ha comentado anteriormente, va más directa a las mesas y sin dar tantos rodeos como hace la arquitectura de Motor Schemas. En conjunto, teniendo en cuenta que se trata de un robot camarero, consideramos más importante la zona de restaurante que la de laberinto y es por esto que elegiríamos la arquitectura subsunción.

7. Problemas encontrados

A la hora de realizar el proyecto, nos han ido surgiendo distintos problemas, que hemos ido solucionando. Para empezar, hemos tenido alguna dificultad con el lenguaje, debido a nuestro bajo conocimiento de *C*. A la hora de familiarizarnos con los distintos ejemplos suministrados acerca de las arquitecturas basadas en el comportamiento, observamos el parpadeo de la luz roja constantemente, no siendo capaces inicialmente de conseguir que dicha luz permaneciera fija.

Al empezar a trabajar con los distintos niveles de competencia, nos aparecía en numerosas ocasiones el error conocido como "Violación de segmento", debido a una mala implementación de los comportamientos diseñados. Al resolver el laberinto inicial antes de llegar a la zona del restaurante, una de las arquitecturas implementadas no era capaz de resolver el mapa planteado inicialmente, por lo que fue preciso realizar una pequeña modificación en la arena.

Además, la batería del robot no se descargaba lo suficiente, hasta que conseguimos entender correctamente el funcionamiento de los distintos coeficientes de carga y descarga. También nos encontramos distintos problemas a la hora de que el robot entregará los pedidos en el restaurante, ya que inicialmente solo atendía las mesas con luz azul más cercanas a la barra del restaurante. Dicho problema, se solucionó jugando con los alcances de los sensores de las luces azules.

Finalmente, el último problema encontrado está relacionado con la manera de encender todas las luces azules, una vez que el robot ha atendido todas las mesas, solucionandolo programando en el *GetTiming* del objeto *blueLightObject* que las luces se encendieran cada 1000 timesteps, ya que comprobamos que para ese tiempo el robot atendía a todas las mesas.

8. Futuras implementaciones

Una vez comprobado el funcionamiento de nuestro robot, se nos ocurren multitud de mejoras para implementar en nuestro proyecto en un futuro, y que por razones de tiempo y complejidad no hemos podido llevar a cabo.

Para empezar, nos hubiera gustado que el número de mesas que existen en el restaurante variara con el tiempo, del mismo modo poder aleatorizar el parpadeo de las luces azules, de tal forma que se enciendan de forma independiente.

Por una parte, en relación al laberinto, podríamos implementar un laberinto más complejo y que nuestro robot se adaptara al igual que hace ahora, eligiendo el camino óptimo.

Finalmente, podríamos experimentar con el número de robots en la arena, de tal forma que cada "X" tiempo el robot que está sirviendo salga de la zona del restaurante recorriendo el laberinto en sentido contrario y al llegar a la salida le pase el testigo a un nuevo robot que empezaría su turno de trabajo. En esta misma línea, podrían estar varios robots trabajando en el restaurante de forma simultanea y que cada uno de ellos se encargue de una zona distinta.

9. Conclusión

A través de este trabajo hemos podido comprobar, las distintas diferencias entre la arquitectura de subsunción y Motor Schemas, en un robot para realizar las tareas de un camarero en un restaurante. Aunque la sociedad está avanzando a un nivel muy rápido, actualmente existen pocos robots camareros en el mundo y especialmente en España, debido a su complejidad para realizar esta función. No obstante, los robots son el futuro, y en un futuro no muy lejano, podremos ver robots entregando pedidos en restaurantes, que les permitirá ofrecer una mayor rapidez en el servicio, aunque todavía tienen bastante que mejorar en el tema de la atención personalizada al cliente.

Para finalizar, destacar que esta tecnología puede ser utilizada en la situación tan difícil que estamos viviendo en la actualidad, por el COVID-19, ya que en China ha sido utilizado este tipo de robots durante la cuarentena para llevar la comida a las personas aisladas y evitar nuevos contagios (Figura 16).

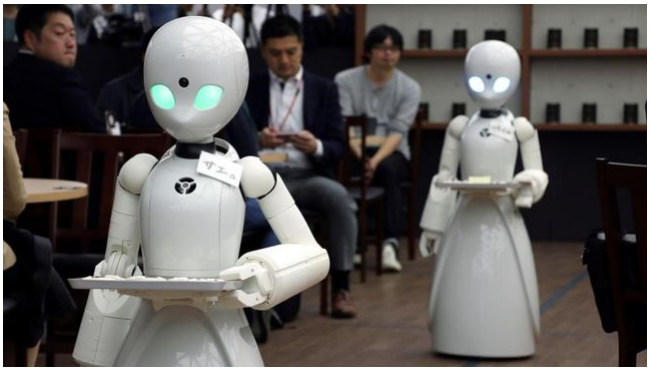


Figura 15: OriHime-D robot camarero en Tokio

Figura 16: Robot Peanut utilizado en China durante el COVID-19

10. Bibliografía

- http://robolabo.etsit.upm.es/subjects.php?subj=irin&tab=tab3&lang=es
- Manual de la asignatura de Introducción a la Robótica Inteligente.
- https://pandas.pydata.org/
- https://seaborn.pydata.org/
- https://matplotlib.org/
- https://www.europapress.es/portaltic/sector/noticia-robot-peanut-lleva-comida-gente-cuare
  html
- https://www.efe.com/efe/america/tecnologia/orihime-d-un-robot-camarero-manejado-a-distanc
  20000036-3824314