TRABAJO 1 IRIN

Jiménez Alonso, Guillermo
guillermo.jimalonso@alumnos.upm.es

Soria Gómez, Alicia
alicia.soria.gomez@alumnos.upm.es

Jarabo Peñas, Alejandro
a.jarabo@alumnos.upm.es

Enlace a GitHub de nuestro proyecto
$https://github.com/alejp1998/irin\_o1$

9 de abril de 2019

Índice

1. Introducción

Queremos (y hemos conseguido) crear un robot cuyo comportamiento corresponda a una arquitectura híbrida, es decir, tenga parte de reconocimiento y cálculo de rutas, y otra parte de simplemente navegación guiada por sensores.

2. Idea

Figura 1: Idea

Nuestra idea consiste en una tarde cualquiera de un joven típico que se va de fiesta con sus amigos. Así pretendemos demostrar que el comportamiento de una persona puede ser simulado por el de un robot, de esta forma, motivando una discusión sobre hasta qué punto las personas somos inteligentes, o actuamos de manera impulsiva o totalmente predecible.

3. Enunciado

Figura 2: Eva

El problema que hemos diseñado se puede resumir con el siguiente enunciado:

El robot Eva tiene una quedada mañana con Walle para ir a la discoteca, no se sabe el camino así que se da una vuelta por la zona siguiendo sus instintos

para aprenderse la ubicación de la casa de Walle y de la discoteca para no llegar tarde mañana, además de la de su casa.

Eva vuelve a su casa, y se duerme. Al despertar se sabe la ubicación de Walle y la disco, así que calcula el camino a casa de Walle y lo sigue con gran seguridad y conocimiento. Cuando llega allí lo recoge y calcula el camino a la discoteca, que también hace sin dudar gracias a su conocimiento.

Mientras están en la discoteca bailan, es decir, se les agotan las pilas, y se emborrachan, así que deciden volver a casa, lo que hacen a trompicones y sin tener ni idea dónde están. Walle dormirá en casa de Eva ya que sus padres son muy estrictos y no le dejan volver tarde a casa.

4. Mapa

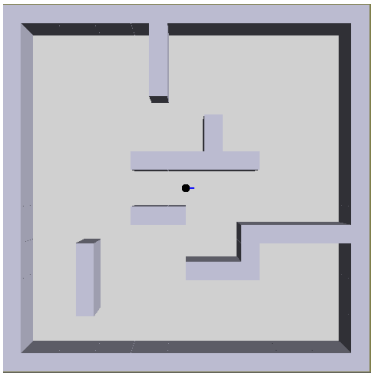

Figura 3: Map1

El mapa básico tiene estas paredes como obstáculos, luego lo hemos rellenado con otros objetos en el paramFiles y ha quedado así:

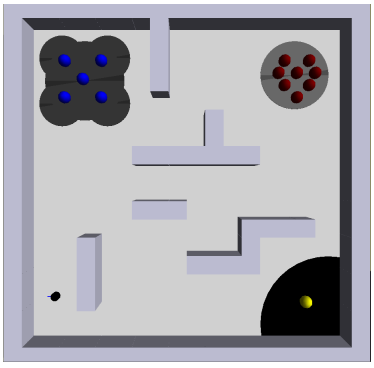

Figura 4: Map1 con Params

Hemos diseñado un mapa tal que la casa de Walle es un corazón con luces rojas sobre un fondo gris, la discoteca tiene muchas luces azules y la casa de Eva tiene una luz amarilla como las que están en las mesillas de noche.

Luego hemos comprobado con más mapas para asegurarnos de la generalidad y corrección de nuestro código y comprobar que no es único para nuestro caso particular.

5. Desarrollo

5.1. Parte de reconocimiento

La primera vuelta que realiza el robot Eva es de reconocimiento, haciendo uso del método BuildMap.

Esto quiere decir que se guía por un comportamiento que hemos programado basado en los datos que obtiene de los sensores (luz roja, luz azul, suelo gris con objetos, luz amarilla).

Siguiendo sus instintos debe crear un mapa que le sirva para calcular rutas y prescindir de sus sensores. El mapa se guarda en la variable Map del método BuildMap y la podemos ver en la consola según se va creando.

Al principio, todo son obstáculos que se ven como O (óes mayúsculas) pero que el robot entiende como "1". Según va avanzando, las casillas por las que sí es capaz de pasar las marca con un . (punto), que él entiende como un "0".

Método Go4Walle Para encontrar la casa de Walle tiene que acercarse a la luz roja y cuando pase sobre el suelo gris que hemos colocado estratégicamente debajo de las luces rojas, se aprende la ubicación Walle y "coge un objeto", esta metáfora tendrá más sentido cuando recoja a Walle pero ahora nos vale para continuar con la vuelta de reconocimiento.

Método Forage Quiere encontrar la discoteca, así que sigue sus sensores de luz azul y se acerca a la zona de luces azules (disco). Cuando llega se aprende la ubicación Disco y se pone a dar vueltas hasta que se le termina la batería, de nuevo esta metáfora tiene más sentido en el contexto de la fiesta, pero de momento es necesario ya que está siguiendo sus sensores.

Método GoLoad Finalmente, con la batería al 0.2 empieza a guiarse por sus sensores de luz amarilla y vuelve a su casa. En el momento en el que el nivel de luz baja drásticamente es porque Eva está debajo de la luz amarilla y los sensores de luz no la están viendo directamente. Aquí se aprende la ubicación de CASA y para de ejecutar el método BuildMap.

5.2. Parte de conocimiento

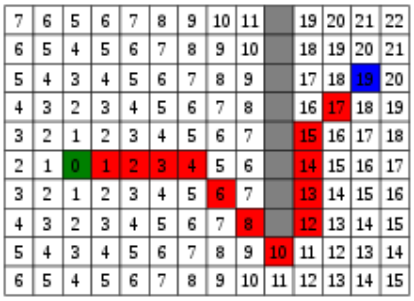

Figura 5: Algoritmo Estrella

En este estado Eva utiliza el mapa que ha construido con BuildMap para calcular una ruta primero desde Home a Walle (camino=0), y luego otra de Walle a Disco(camino=1) con el método PathPlanning que utiliza el método PathFind.

5.2.1. Girando

Debemos colocar a Eva con un ángulo de pi radianes para que se lleve a cabo de manera adecuada el camino. Cuando este ya esté situado, se pone la posición calculada de casa como casilla de comienzo, y la casa de Walle como casilla de final, y se calcula la ruta a seguir con PathPlanning, además de poner el booleano starEnd a false.

5.2.2. Primera ruta

Se calcula la ruta entre Home y Walle. La variable camino está a 0. Cuando llega al destino, es decir, cuando HybridManager ve que su estado es mayor que el número de estados planeado, lo indica poniendo camino = 1 y se prepara para la segunda ruta (gira hasta apuntar a pi radianes).

5.2.3. Segunda ruta

Con la variable camino=1, se calcula la ruta con inicio Walle y destino Disco, y cuando llega al destino pone starEnd=true, lo que indica que la arquitectura de conocimiento ha terminado y aplicamos la metáfora de que ambos están borrachos y bailando.

5.3. Parte del comportamiento sensorial

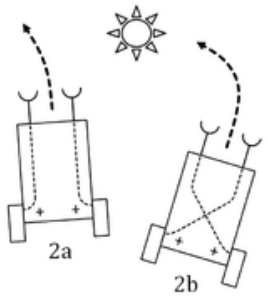

Figura 6: Vehiculo de Braitenberg

La mejor analogía que se nos ha ocurrido para una persona (o robot) que no es consciente de dónde está ni lo que debe hacer, es emborracharlo, así que eso hemos hecho. Desde que Eva abandona el comportamiento de conocimento y la consciencia responsable, se pasa al comportamiento instintivo y sensorial. Así, baila hasta agotarse y vuelve a casa sin casi enterarse de nada.

Bailando/Método Forage el estado del robot en el que sigue a sus sensores de luces azules y se le está agotando la batería lo hemos asociado al baile (nos ha parecido bastante adecuado). Dura hasta que la batería está al 0.2 y entonces empieza el método GoLoad para volver a casa. Según se va alejando de las luces azules se le va a agotando la batería así que en realidad su mínimo de batería llega a ser bastante menor que 0.2.

GoLoad y vuelta a casa como hemos programado, Eva lee los datos de los sensores de luces amarillas y se acerca a ellas para 'irse a dormir', y terminar así el comportamiento que hemos querido imitar de un@ joven de fiesta. Cuando llega a su cama (bajo la luz), apaga la luz de su mesilla de noche y se duerme(color verde) mientras recarga energía; hasta que se despierta, apaga la luz y vuelve a la rutina.

5.4. Comportamiento Híbrido

Todos estos comportamientos no ocurren a la vez, sino que se van sucediendo unos a otros según el método HybridManager. Este se ejecuta cada SimulationStep para decidir qué comportamiento debe adoptar el robot, además de gestionar en que momento alterna entre estos según la información de sus sensores.

5.5. Continuación

Este programa no está diseñado para terminar, y a Eva no se le olvida el mapa. El programa continuaría eternamente de la siguiente manera (que ya hemos explicado anteriormente) :

1. Ruta consciente de Home a Walle
2. Ruta consciente de Walle a Disco
3. Ruta instintiva de Disco a Home

6. Nuestras variaciones

6.1. bluelightobject.cpp

Dentro de objects. Modificamos la rapidez de encendido modificando el periodo un factor de 100 con motivos puramente estéticos.

6.2. redlightobject.cpp

Las luces rojas estaban prediseñadas para parpadear muy lentamente, debido a esto, el robot no era capaz de situar correctamente la posición de la casa de Walle (totalRedLight se pone a 0.0 cuando están apagadas). Lo hemos solucionado eliminando dicho parpadeo, comentando el interior del método que lo produce.

6.3. batterysensor.cpp

Dentro de sensors. Añadimos un estado en el que la batería se descargue rápidamente si está suficientemente cerca a la luz azul utilizando la variable mfBatteryLevel -= mfDischargeCoef*3. Hemos eliminado el estado en el que se carga dentro del rango de cercanía de la luz amarilla, sustituyéndolo por un método DrunkSleep para cargar que llamamos cuando los sensores de luz amarilla están a cero, es decir, cuando apagamos la luz después de volver de la fiesta.

6.4. batterysensor.h

Dentro de sensors. Añadimos el método de DrunkSleep para que esté declarado.

6.5. iri1controller.h

Dentro de controllers. Añadimos los métodos creados en el .cpp para que estén declarados.

6.6. iri1exp.cpp

Dentro de experiments. Aquí se crea el mapa como un array de almohadillas como suelo y porcentajes, como paredes, no se crean las luces ni el suelo, estas van en el paramFiles.

6.7. O1Map1.txt

Dentro de paramFiles. Declaramos el resto de objetos que queremos que aparezcan en el mapa, en particular el primer mapa que hemos hecho es:

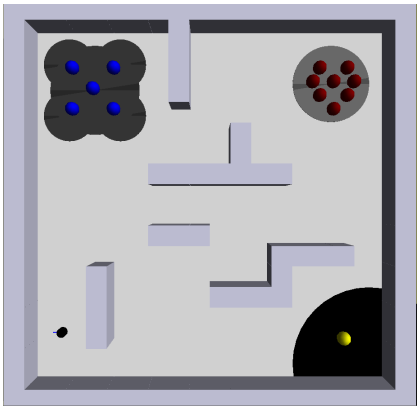

Figura 7: Map1 with Params.

7. iri1controller.cpp

Es el archivo más importante. Nos hemos inspirado en el archivo subido a Robolabo "Ejemplos de Arquitecturas basadas en el Comportamiento" en particular, motorschemas2controller.cpp, y también en los "Ejemplos de Arquitecturas basadas en el Conocimiento", en particular el map5controller.cpp.

Se declaran todas las variables globales que utilizaremos en los métodos, los nombres son bastante explicativos y hay comentarios detallando lo es cada variable. Nos centraremos por tanto en esta memoria en explicar lo que hacen los métodos que hemos adaptado a nuestro caso particular y los que hemos creado nosotros.

7.1. Estados

[1] Tenemos 5 estados por los que pasa el robot a lo largo de su vida, y tienen varias importancias. Por ejemplo, si el robot se queda sin batería mientras está en Forage, se activa (=0) la variable de inhibición de Forage y pasa a Reload. Lo mismo ocurre si está yendo a por batería y se choca con algo, automáticamente se activa (=0) la inhibición de Reload en favor de Avoid para que no se choque. Ambos inhibidores están desactivados al principio debido a ExecuteBehaviors.

Además existe una tabla ActivationTable que guarda en la columna [0] el ángulo asociado al movimiento de un estado particular, en la columna [1] el dato maś fuerte que recogen los sensores adecuados a ese estado y en la columna [2] se indica si ese estado está activado, los estados comienzan desactivados debido a ExecuteBehaviors.

Navigate: se encarga de que el robot esté en movimiento siempre por defecto, no influye en la decisión del ángulo de giro ni en el dato más fuerte detectado, y está siempre activo.

Avoid: es el comportamiento con mayor prioridad de todos (al activarse inhibe a todos los demás), se encarga de evitar chocar contra las paredes calculando un ángulo de escape cuando sus sensores de proximidad estén por encima de un determinado umbral.

Forage, Go4Walle, GoLoad: son los comportamientos encargados de dirigir al robot hacia un objetivo (luces azules, rojas y amarillas respectivamente). Se activan en los siguientes casos:

Forage: cuando el robot ha cogido un objeto (groundMemory[0] = 1), se dirige hacia las luces azules.

Go4Walle: mientras que el robot no haya cogido un objeto (groundMemory[0] = 0), se dirigirá hacia las luces rojas.

[1] AVOID, RELOAD, FORAGE, NAVIGATE y GO4WALLE

GoLoad: cuando la batería esté por debajo del umbral, entonces inhibirá a los dos comportamientos anteriores, y se dirigirá a su casa a recargar batería.

Cuando un estado se activa, acudimos al método Coordinator: a este se le pasa la información del ángulo guardado en la tabla a las ruedas para seguir una dirección en particular. Es por así decirlo, el método más físico de todos, el que realmente mueve las ruedas.

7.2. Secuencia de métodos

Existe una secuencia de métodos que merece la pena comentarlos de manera ordenada ya que la salidas/ acciones de uno repercuten directamente en el siguiente:

7.2.1. BuildMap

El método BuildMap es esencial para conseguir la arquitectura híbrida mezclando conocimiento y comportamiento sensorial. Este es el método que se está corriendo durante la vuelta de reconocimiento inicial y mientras el robot vuelve a casa por estímulos. En él, guardamos en la variable map[][] las casillas rellenas de obstáculos "1" que vemos como Oes, y las casillas libres "0" que vemos como . (puntos). Además nos permite guardar la ubicación de la Home, de Walle y de la Disco que utilizaremos más tarde como salida y destino de las dos rutas que calcularemos mediante PathPlanning().

7.2.2. PathFind

Cuando BuildMap ya tiene su mapa que ha ido descubriendo, el método PathPlanning() aplica el algoritmo Estrella mediante pathFind, similar a un Dijkstra, ya que va encontrando los nodos a recorrer entre un origen y un destino y te devuelve un string "path" que es el camino libre de obstáculos entre origen y destino.

7.2.3. PathPlanning

Con la rutas que nos devuelve PathFind, aplicamos el método PathPlanning que se encarga de conectar dichas rutas y cuando finalice imprimir el camino calculado por pantalla.

7.2.4. GoGoal

GoGoal simplemente calcula cómo moverse entre una parada y otra para poder seguir correctamente el camino de PathFind. Mientras CalcPositionAndOrientation actualiza y normaliza la orientación y posición del robot, para así determinar si se ha llegado con éxito a las paradas previamente determminadas con PathPlanning().

7.3. Métodos Subsunción

Aquí tenemos los métodos asociados a los estados:

7.3.1. ObstacleAvoidance

ObstacleAvoidance es sencillo. Coge la información de los sensores de proximidad y crea un ángulo repelente que guarda en ActivationTable. Solo se activa este estado en el caso de que fMaxLight que es el sensor que más siente por así decirlo, sea más fuerte que un umbral definido por nosotros para que solo evite choques cuando esté muy cerca de una pared. Al activarse también inhibe el estado de GoLoad, Go4Walle y Forage con la variable de fAvoidToBattInhibitor=0. Esto se hace para evitar que al calcular la media entre ambos ángulos acabe chocando contra la pared, y llegue (tarde o temprano a sus objetivos).

7.3.2. GoLoad

Cuando la batería del robot baje de un determinado umbral, el método GoLoad produce un ángulo de atracción tomando la información de los sensores de luz y la guarda en la ActivationTable, activando también el estado GoLoad en la tercera columna de la ActivationTable. Hemos creado una condición en la que si el robot está muy cerca de la luz, el umbral de batería que no te deja salir de Reload sea muy alto para que se cargue bien porque sino en cuanto tuviese más del 20/100 de batería (umbral en el que entra al estado Reload), saldría del estado Reload. Podemos entenderlo como un estado de histéresis forzado, en el que dos umbrales determinan la entrada o salida de un estado (Reload)

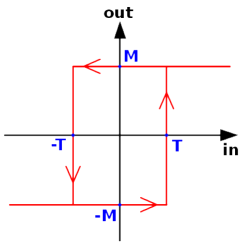

Figura 8: Histéresis

7.3.3. Go4Walle

Es el método asociado a las luces rojas, se activa cuando el robot no tiene un objeto (groundMemory[0] = 0). De forma análoga a GoLoad calcula un ángulo de aproximación según la información que reciba de sus sensores de luz roja, que guarda en fAngle y almacena el mayor valor que haya obtenido de sus sensores en fMax.

7.3.4. Forage

Es el método asociado a las luces azules que tiene que activarse después de haber recogido a Walle, es decir, cuando la memoria del suelo esté activada(groundMemory[0] = 1). Actúa como Go4Walle, hasta que el robot llega a las luces azules y comienza a descargarse mientras "baila".

7.3.5. Navigate

es el método de "reserva" como hemos querido llamarlo. No introduce ningún ángulo de repulsión o atracción así que sirve para que el robot, a falta de estímulos, se siga moviendo hacia delante.

7.4. Hybrid Manager

Finalmente tenemos el método que se ejecuta cada paso, es decir cada vez que pulsas CTRL+o, o que se ejecuta automáticamente cuando pulsas CTRL+p. Este es el SimulationStep que recoge datos de los sensores de luz, posición, proximidad y orientación y se los pasa al HybridManager, el método que guía al robot. El HybridManager define varios estados dentro de otros estados, lo estableceremos con una tabla para que se vea con mayor facilidad.

Figura 9: HybridManager

Las dos más generales son !starEnd o starEnd, que indican si estamos siguiendo un comportamiento de conocimiento o de impulsos, respectivamente.

7.4.1. Conocimiento

Si estamos en el conocimiento podemos haber terminado una ruta o seguir en ella.

1. Si seguimos en ella no ocurre nada y simplemente seguimos sumando mnState++ cada vez que la posición calculada esté dentro de un determinado

error respecto de la posición de la siguiente parada a la que pretendía llegar.

2. Si ya hemos terminado la ruta (mnState ¿= mnPathPlanningStops) podemos o bien : i) haber terminado la primera (buscar a Walle) y entonces nos prepararemos para entrar en la segunda o bien ii) hemos terminado la segunda (ir a disco) y entonces salimos del conocimiento y nos metemos en los impulsos (metáfora del borracho).

7.4.2. Impulsos

Si estamos en los impulsos solamente podemos estar en reconocimiento y actualizando el mapa (incluso cuando está borracho, como sigue la arquitectura de subsunción también está rellenando el mapa de Oes y puntos).

7.4.3. Vuelta a casa

Una vez lleguemos a la cama, estaremos debajo de la luz, por lo que la lectura de el total de los sensores de luz amarilla bajará drásticamente a un valor por debajo de uno, y además la luz azul detectada será bastante menor a la amarilla (está en la otra esquina). En ese momento la luz se apagará y el robot comenzará a girar mientras dicha luz esté apagada (sensores luz amarilla a cero) hasta que la batería esté por encima del 0.9 y además esté orientado hacia pi radianes. Cuando ocurre esto, entonces se calcula la ruta hasta la casa de Walle mediante PathPlanning() y se enciende la luz de nuevo; además de poner el booleano starEnd a false para indicar que seguiremos el comportamiento basado en el conocimiento.

8. Gráficas

Para realizar los gráficos usamos los ficheros generados al iniciar el trabajo en el simulador, y *python*, haciendo uso de sus librerías. *Pandas* para el tratamiento de estos ficheros y convertirlos en *dataframes* o especie de matrices para trabajar más facilmente con los datos, *matplotlib* para generar los gráficos a partir de estos *dataframes*, *numpy* para manipulación de arrays y *seaborn* para otras visualizaciones.

Los ficheros de texto plano que hayamos usado y que se crean en la carpeta *outputFiles*, tanto los originales como los creados por nosotros son: *avoidOutput*, *batteryOutput*, *behaviorOutput*, *robotPosition*, *sensorLuzOutput*.

En *behaviorOutput* lo que se genera son filas con las variables de tiempo y parsi estan activados o no cada comportamiento (flag de activacion) y de *sensorLuzOutput* se relaciona el tiempo y las variables *totalLight*, *totalBlueLight* y *totalRedLight*, que guardan la suma de los valores que reciben los respectivos sensores en cada timestep.

A continuación se presentan los gráficos.

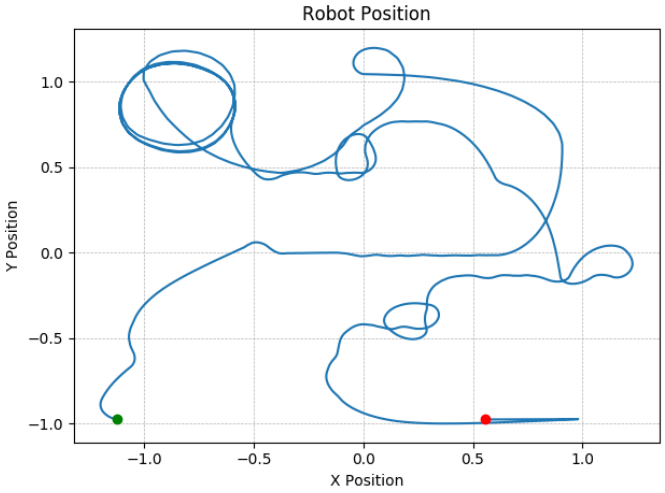

Figura 10: Trayectoria del robot

En este gráfico se observa la trayectoria del robot, comenzando en el punto verde, y acabando en el punto rojo. Tanto para esta visualización como para las siguientes solo se ha tenido en cuenta la vuelta de reconocimiento pues para las demás vueltas la información recogida será la misma o parecida.

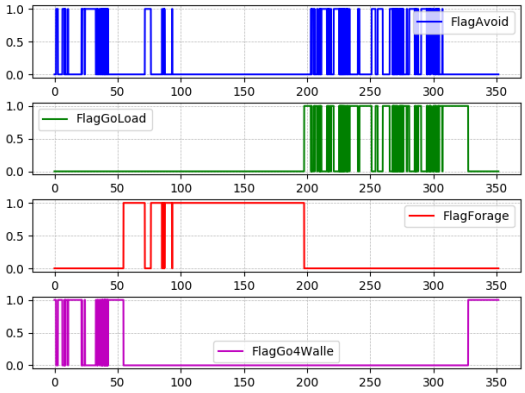

Figura 11: Activación de comportamientos

En la representación de arriba se observa cómo se van intercambiando la activación de los distintos comportamientos según el tiempo (timesteps). Se ha omitido el comportamiento *Navigate* pues su flag está siempre a 1 y carece de información. En el gráfico de abajo se han calculado el total de timesteps en los que cada comportamiento está activado, siendo el predominante *Forage* debido a que Eva pasa mucho tiempo en la discoteca.

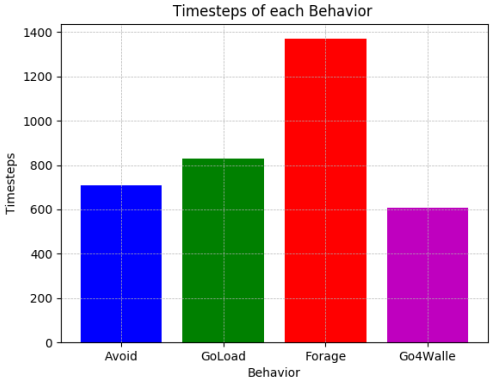

Figura 12: Timesteps por comportamientos

También se puede asociar la posición al comportamiento que tiene en ese momento el robot como queda patente en la trayectoria del dibujo de la página siguiente. Los tramos más cortos son los azules, que hacen referencia a *Avoid*, pues el robot tiene que maniobrar para no chocarse y implican la existencia de obstáculos cercanos. Se observa también como comienza en *Go4Walle*, después cambia a *Forage* donde empieza a dar vueltas y finalmente se le acaba la batería y se dirige a la luz amarilla con *GoLoad*.

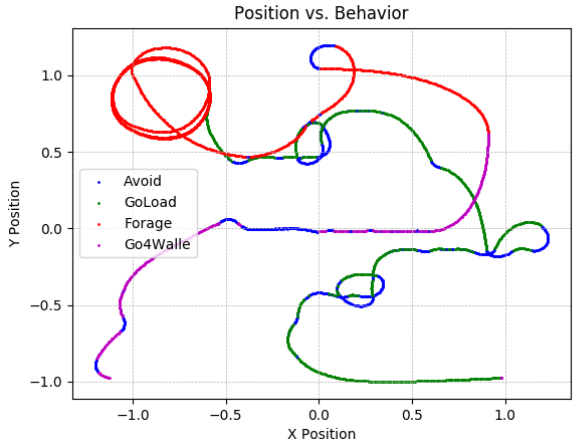

Figura 13: Comportamiento en el recorrido

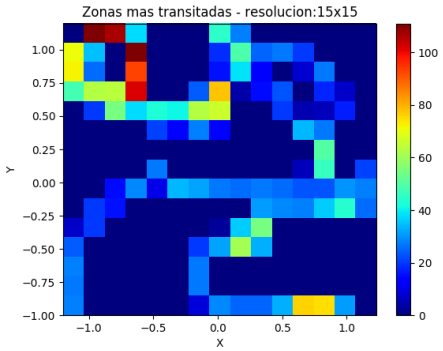

Figura 14

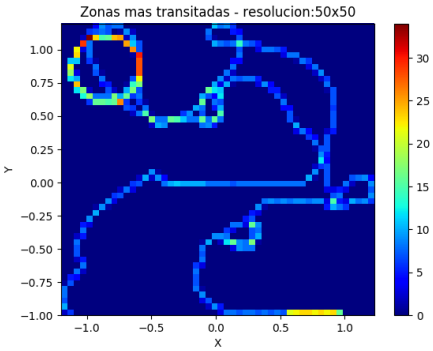

Figura 15

Las gráficas de arriba representan mapas de calor en distintas resoluciones mostrando las zonas más frecuentadas por el robot. Realmente es el recorrido que sigue Eva, porque dado un mapa, tiene un camino para sus propósitos y

no va vagando como tal, por eso no es una representación completa. De todas formas se observa que en la discoteca está mucho tiempo, pasando varias veces por las mismas zonas, asi como en su casa, la luz amarilla.

En la siguiente imagen se visualiza la misma información mediante las curvas de nivel. Estas significan de alguna manera el rango de acción del robot, por donde se centra su movimiento. Cuanto más rojas y juntas sean las curvas, más tiempo y veces pasa por esas posiciones.

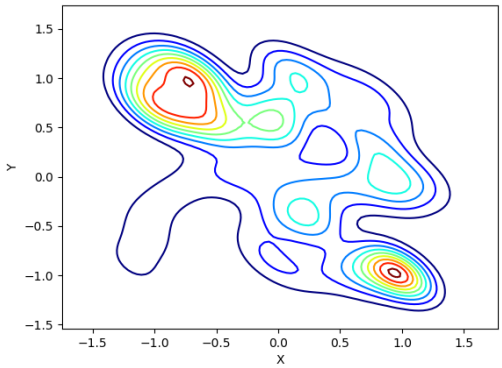

Figura 16: Curvas de nivel por posición

Unos de los sensores más utilizados son los de las luces amarilla, azul y roja. Según la información recopilada por ellos y asociándolos a la posición en esos instantes tenemos las siguientes gráficas. De ellas se puede extraer que efectivamente hay alta probabilidad de tener focos de luces amarillas en la esquina inferior derecha, luces azules en la esquina superior izquierda y luces rojas en la esquina superior derecha como muestran los codigos de colores normalizados.

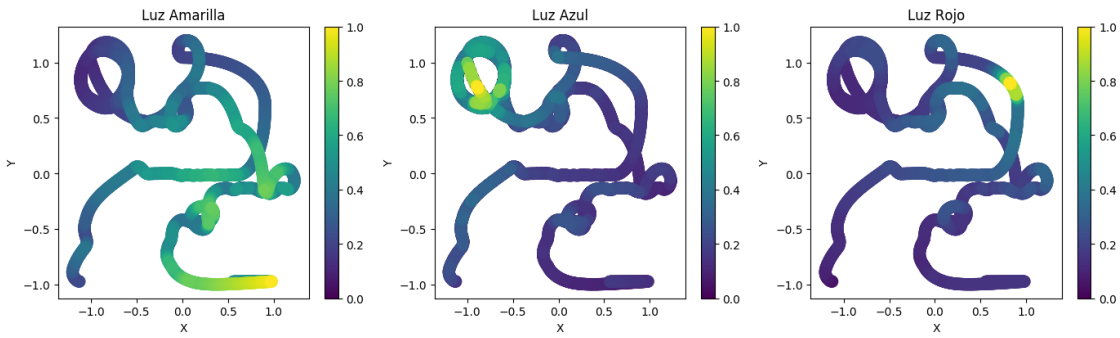

Figura 17: Mapas de calor según el sensor de luz

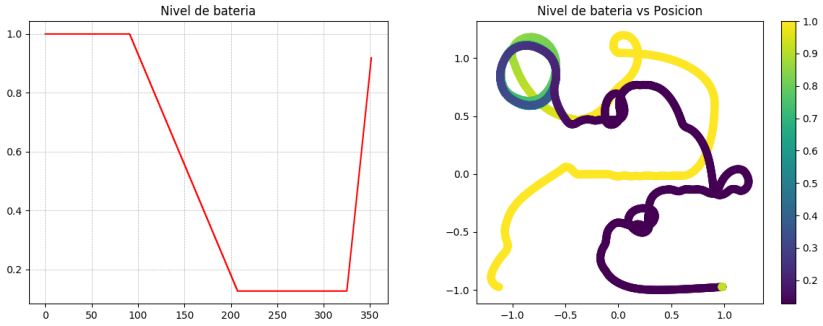

Figura 18: Nivel de batería

Por último tenemos el nivel de bateria según el tiempo, y su nivel según la posición en la que se encuentra el robot como muestras las anteriores imagénes.

9. Conclusiones

Figura 19: Walle

Deducimos que no podemos comparar una persona a un robot programado. No solo por la cantidad de variables que la persona tiene en cuenta y que sí se podrían implementar en el robot, sino por un factor psicológico del que no podemos dotar al robot. En la mayor parte de casos ni siquiera nosotros somos capaces de explicar el por qué de nuestras acciones. Por ejemplo, ¿qué pasaría si Walle y Eva hubiesen discutido? ¿hubiese ido ella a buscarle? ¿tendría sentido siquiera este proyecto? Hay múltiples factores internos a las personas (culturales, de educación, personalidad, relación con otras personas, ...) que no podemos asociar a un robot. ¿Tendría nuestro robot Eva capacidad de perdonar a Walle?

¿O de enfadarse con él? ¿Y si deja de caerle bien? Todos estos factores no los podemos encontrar en un robot, y es lo que hace que al final nos distingamos de ellos, por muy 'inteligente' que pueda parecer su comportamiento.

10. Problemas encontrados

Durante la resolución de este trabajo nos hemos ido enfrentando a diversos problemas. La mayoría han sido relacionados con dificultad al entender el código ya existente y al intentar implementar nosotros funciones. Esto se debe a que tenemos una base de programación muy amplia pero poco detallada, así que diseñábamos métodos que estaban bien planteados pero en los que la sintaxis hacía referencia a otro objeto y casos similares. Por suerte conseguimos solucionarlos todos a tiempo.

11. Futuras extensiones

11.1. O1Map2.txt

Los objetos están dentro del .tx dentro de paramFiles y las paredes las definimos en el iri1exp.cpp. Este es el segundo mapa que hemos diseñado y en el que el código funciona, no supone mayor dificultad pero comprobamos que el código es general y correcto y que no está limitado a nuestro caso particular. Ha quedado así:

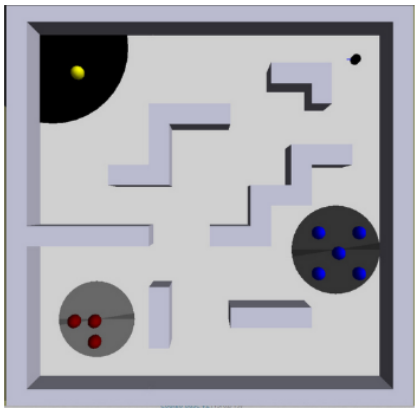

Figura 20: Map2

Para este mapa también sacamos los mismos gráficos que para el anterior y a continuación pondremos los más relevantes.

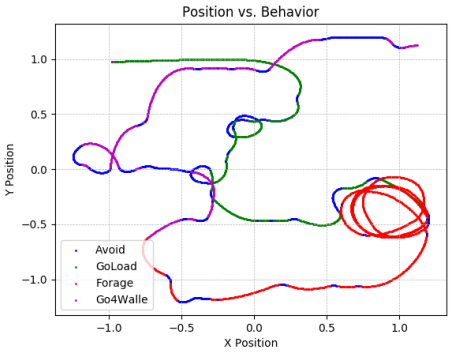

Figura 21: Comportamiento en el recorrido

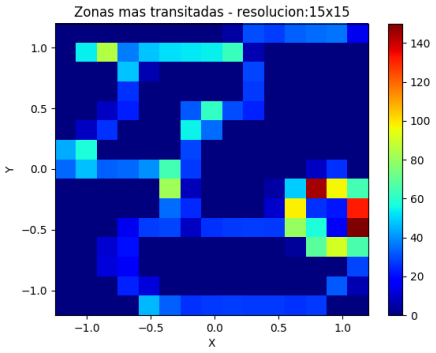

Figura 22: Mapa de calor por posición

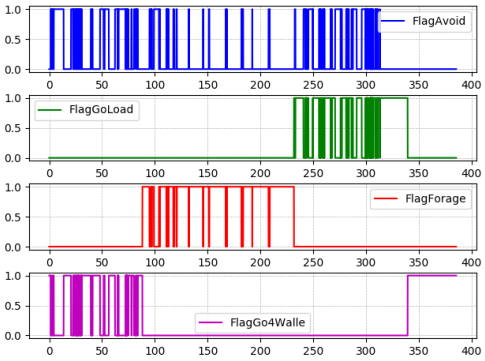

Figura 23: Activación de comportamiento

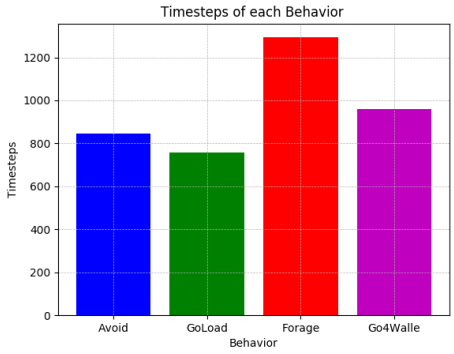

Figura 24: Timesteps por comportamiento

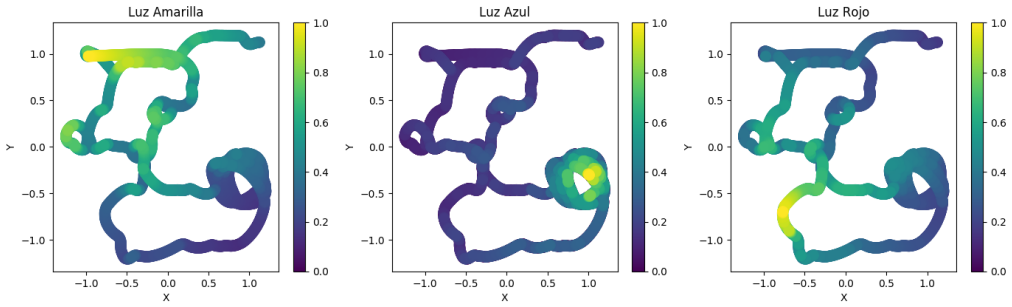

Figura 25: Mapas de calor según el sensor de luz

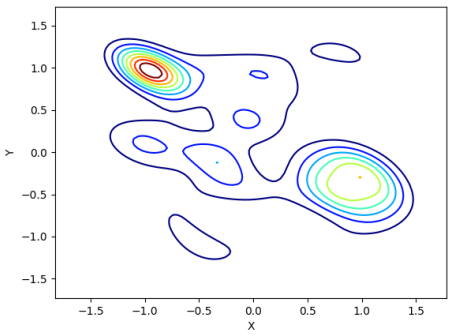

Figura 26: Curvas de nivel por posición

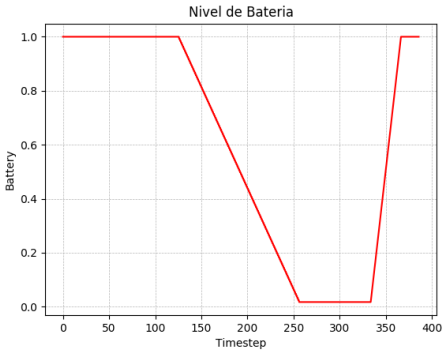

Figura 27: Nivel de batería

11.2. Ideas hacia el futuro

Al comienzo del trabajo, tuvimos que pensar en alguna idea que diseñar y se nos ocurrieron múltiples ideas interesantes que al final no implementamos porque decidimos centrarnos en la que hemos presentado a lo largo de toda esta memoria. De todos modos nos quedamos con la espina clavada de llevar a cabo el resto de propuestas que surgieron en la primera aproximación a este trabajo. Por ejemplo, queríamos darle opción a Walle de darle calabazas a Eva con una variable de tipo random para acercarnos más al comportamiento de las personas (impredecibles en muchos casos), y que ella se volviese a su casa en este caso. Creemos que no sería difícil de implementar con los conocimientos que tenemos actualmente. También pensamos en pasarle varios mapas al programa en iri1exp.cpp y que fuesen saliendo de manera aleatoria, además de asociarlos a un paramFiles determinado, solo que no sabíamos si era posible puesto que es el usuario el que elige el paramFiles a ejecutar cuando introduce el comando ./irsim -E 30 -p paramFiles/laquetuquieras.txt así que lo descartamos.

12. Nuestro pensamiento al respecto

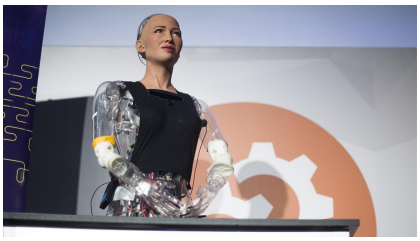

Figura 28: Mujer Robot

Nuestra opinión como grupo de estudiantes de IRIN hacia este trabajo es positiva. Para empezar estamos contentos de haber tenido la libertad de elegir un enunciado que nos resultase atractivo y trabajar sobre él con las herramientas accesibles. Un grupo motivado siempre trabaja mejor que uno sin ganas al que se le asigne un proyecto estándar (lo que ocurre casi siempre en esta escuela).

Además, en el ámbito de la robótica y el comportamiento, hemos aprendido que por muchas variables y estados que programemos en un robot, siempre le faltará el toque humano de emociones personales y cierta irracionalidad instintiva con la que nos caracterizamos las personas. Creemos haber adquirido una visión más amplia de los objetivos que deberíamos tener a la hora de crear o al menos querer crear un ser 'inteligente' y somos mucho más conscientes de las dificultades que esto implica.

13. Bibliografía

1. Advanced Robotics and Intelligent Machines, J. O. Gray, Darwin G. Caldwell, Institution of Electrical Engineers, 1996

2. I, Robot, Isaac Asimov, 1991

3. Handbook of Industrial Robotics, Shimon Y. Nof, 1999

4. https://matplotlib.org/

5. https://pandas.pydata.org/

6. http://www.numpy.org/

7. https://seaborn.pydata.org/

8. https://jakevdp.github.io/PythonDataScienceHandbook/