ESCUELA POLITÉCNICA

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA
INGENIERÍA INFORMÁTICA EN INGENIERÍA DE COMPUTADORES

TRABAJO FIN DE GRADO

Localización de robots en tiempo real y simulación de comportamientos biológicos

UNIVERSIDAD DE EXTREMADURA

ESCUELA POLITÉCNICA
INGENIERÍA INFORMÁTICA EN INGENIERÍA DE COMPUTADORES

TRABAJO FIN DE GRADO

Localización de robots en tiempo real y simulación de comportamientos biológicos

Autor: Ismael Sánchez García
Tutor: Pablo Bustos García de Castro

Resumen

En las últimas décadas hemos visto cómo evoluciona la tecnología y cómo el ámbito tecnológico se va introduciendo poco a poco en todos los sectores, pero sobre todo en el sector primario, uno de los que ha contado siempre con menos tecnologías. Este trabajo cubre una de las fases del proyecto llamado 'Caracterización etológica de corderos con tecnologías robóticas' en el cual trabaja actualmente Robolab en colaboración con EA group S.C. Este empresa ha planteado la hipótesis de que una selección adecuada de animales en los corrales de cebo mejorará el ritmo de engorde y, por tanto, el ratio coste/beneficio. Esta selección estaría relacionada con el *carácter* del animal y la forma en que su comportamiento afecta a otros animales.

El objetivo de este trabajo es crear un ecosistema de robots autónomos virtuales que simulen los comportamientos básicos (andar, comer, beber y dormir) de los corderos y así poder iniciar estudios e investigaciones de tipo etológico. Los modelos creados y la simulación reducirán considerablemente el tiempo de los estudios en comparación con los experimentos realizados en corrales reales.

Como punto de partida para este trabajo se decidió utilizar dos estructuras de control diferentes para el comportamiento de los robots: las máquinas de estados y los árboles de comportamiento. Todo el trabajo se ha realizado con un único robot en el ecosistema. Cuando todo comenzó a funcionar correctamente se añadió un segundo robot para estudiar las interacciones entre ambos. Finalmente, se introdujeron un total de cinco robots para obtener datos de interacciones a una escala más representativa.

Uno de los objetivos del proyecto en el cual está trabajando Robolab utiliza radios con tecnología UWB para la localización en tiempo real de los corderos. Para introducir estos elementos en el modelo que ha sido creado, se ha simulado en los robots virtuales la posición estimada por estos dispotitivos incluyendo el ruido que estas introducen en la medida. Los datos reales del ruido se han obtenido a partir del sistema de radios disponible en el laboratorio.

Con los datos que han sido obtenidos mediante la simulación se han generado diversas gráficas, las cuales muestran el recorrido individual de cada robot. En otra gráfica adicional se muestra el recorrido de todos los robots en un mismo mapa simultáneamente y, por último, se ha generardo un mapa de calor mediante el cual se pueden visualizar las zonas del mapa que han sido más visitada por los robots.

Abstract

In the last decades, we have been witnessing how technologies evolve and how the technological field is getting introduced rapidly in every sector but specially on the primary sector, one of those which always had less technologies on it. This work, that is one of the phases from the project called 'Lambs ethological characterization with robotics technologies' in which Robolab is currently working in cooperation with EA group S.C. This company has suggested the hypothesis that an appropriate selection of animals in the farmyards will improve the rate of fatten the lambs up, therefore, the cost/benefits ratio. This selection would be related to the animals' behavior, and how their behavior affects to other animals.

The goal of this project is to create an ecosystem of virtual robots which simulate the lambs' basics behavior (walk, eat, drink and sleep) and with this information be able to initiate ethological studies and ethological research. The models that have been created and the simulation will considerately reduce the time spent in the research compared to real research made in real farmyards.

As starting point for this work, it was decided to use two different control structures to implement the robots' behavior: state machines; and behavior trees. The whole work has been realized with an only robot in the ecosystem. When everything start working correctly, a second robot was added to study the interactions between both of them. Finally, five robots was introduced in the ecosystem to obtain data from the interactions in a more representative scale.

One of the project's goals in which is working Robolab used radios with UWB technology for the localization in real time of the lambs. To introduce this elements in the model that has been created , the estimated position of the virtual robots has been simulated with this device including the error that exists in this measurements. The data about the error has been obtained from the radio system which is installed in the laboratory.

With the data that have been obtained through the simulation, some graphs have been generated, which represent the robot's individual path. In another additional graph, the robots' paths have been shown on the same map at the same time. Finally, a heat-map have been generated with which it can be seen the area that is more visited by robots.

Índice general

Índice de figuras

Capítulo 1

Introducción

En las ultimas décadas, venimos presenciando como evoluciona internet y las nuevas tecnologías y como estas se van integrando rápidamente en la mayoría de los aspectos de la vida. Durante el desarrollo de este trabajo, se verá como la tecnología se pueden integrar en parte del sector primario. Este trabajo forma parte de un proyecto llamado 'Caracterización etológica de corderos con tecnologías robóticas', en el cual está trabajando actualmente Robolab, un grupo de investigación de la Universidad de Extremadura.

La empresa solicitante de este proyecto es EA group S.C. Es una cooperativa de segundo grado Extremeña, la cual fue creada como un proceso de integración cooperativa dentro del sector ovino de carne. A día de hoy, EA group S.C. es el mayor grupo cooperativo de comercialización de ovino en Europa.

Figura 1.1: EA Group S.C.

Esta cooperativa quiere implementar un sistema de medida que actualmente todavía no existe en el mercado y que presenta nuevos retos tecnológicos para estudiar las condiciones de los cebaderos y la densidad de animales en ellos. Además de esto, también quieren estudiar el comportamiento de los animales agrupados en lotes y el rendimiento final de ello. Esto se debe a que se cree que se puede mejorar el rendimiento organizando a los animales teniendo en cuenta el potencial de engorde ya que actualmente los lotes se organizan por edad o por el peso actual de los animales y esto haría que se reduciera el coste del alimento y del tiempo de alimentación del animal. También se debe a que el comportamiento social de los corderos ya que por el tamaño se generan jerarquías de grupo y esto puede afectar a la mala alimentación de alguno de los animales.

EA group S.C comercializa alrededor de setecientos mil corderos al año. Para la empresa, el kilogramo de carne repuesto, le cuesta entre uno con veinte y uno con cuarenta euros por kilogramo, dependiendo del precio del alimento, y se encargan de cebar a los corderos aproximadamente seis kilogramos, por lo que conseguir disminuir el coste del alimento o el tiempo de alimentación del animal es una gran ventaja económica para esta.

Para ello, este proyecto consta de varias fases de estudio e investigación, mediante robots reales y robots virtuales. Para poder llegar a este objetivo es necesario realizar estudios con robots virtuales, es decir, simulaciones. Esto es necesario para reducir los tiempos y el coste que supondría la experimentación en el cebadero.

Esta parte del proyecto será la que se desarrolle este trabajo, donde se creará un ecosistema de robots autónomos virtuales en el cual se simularán los comportamientos más básicos de los corderos como son andar, comer, beber y dormir. Además estarán monotorizados en todo momento obteniendo las coordenadas por las que se mueven por el mapa virtual, y así poder estudiar sus rutinas.

Capítulo 2

Objetivo general del proyecto

El objetivo principal es conseguir crear un ecosistema de robots virtuales autónomos los cuales puedan simular el comportamiento biológico básico de los corderos y que estos publiquen cada cierto tiempo sus coordenadas en las que se encuentran en tiempo real, para poder realizar un análisis de los datos obtenidos. Para conseguir este objetivo, hay que realizar varias tareas, que se definirán a continuación:

- Conseguir que un robot realice las cuatro tareas básicas del comportamiento de los corderos (andar, comer, beber y dormir) mediante el modelo de la máquina de estados.
- Una vez que el robot esté funcionando correctamente mediante la máquina de estados, investigar y estudiar el funcionamiento de los árboles de comportamiento.
- Implementar el funcionamiento que se ha conseguido recrear mediante la máquina de estados en un árbol de comportamiento.
- Una vez que el robot esté funcionando mediante el árbol de comportamiento, se añadirá un nuevo robot a la ejecución.
- Cuando el segundo robot esté implementado, se estudiarán los errores que se producen durante la interacción de ambos robots.

- Tras corregir los errores de interacción encontrados entre ambos, se implementará para las funciones de Andar, IrComer e IrBeber, un método en el cual si dos robots se encuentran, se esquiven y no acaben cruzandose.
- Cuando los dos robots estén funcionando correctamente, se añadirán los tres robots restantes para realizar la ejecución del estudio con cinco robots.
- Una vez se encuentre todo funcionando correctamente, se simulará el ruido generado por las radios MDEK1001, para introducirlo en la simulación.
- Cuando la simulación del ruido generado de las radios esté implementado, se creará la función de publicación de la posición de los robots en tiempo real en ficheros de texto.

Capítulo 3

Estado de la técnica

3.1. Lenguajes de programación utilizados

Para la realización de este proyecto se han usado distintos lenguajes de programación y librerías que se explicarán en este apartado.

3.1.1. C++

'C++' es un lenguaje de programación creado con la intención de ampliar el lenguaje de programacion 'C' con funciones y características que permiten la manipulación de objetos. Las principales características de este lenguaje son:

- Es un lenguaje multiparadigma ya que soporta más de un paradigma de programación entre los que se encuentran:
    - Orientado a objetos: paradigma de programacion en el cual el código se organiza en unidades que reciben el nombre de clases de las cuales se instancian objetos los cuales se relacionan entre si con el fin de conseguir los objetivos del programa. Cada objeto ofrece una funcionalidad específica la cual trata los datos de entrada para obtener los datos de salida deseados. Los principales pilares de la programación orientada a objetos son:

- Encapsulación: agrupar los elementos pertenecientes a cada entidad al mismo nivel de abstracción de manera que se consigue aumentar la cohesión del sistema.
- Herencia: los objetos pueden formaruna jerarquía de clasificación con la cual es posible que un objeto herede las propiedades y métodos de la clase a la que pertenece. Esto favorece la reutilización de código y permite a los objetos ser creados a partir de la definición de objetos ya existentes. Aunque un objeto herede las características de la clase superior, este podrá definir sus propios métodos y variables a la vez que modificar el comportamiento de los métodos heradados en algunos casos.
- Polimorfismo: es la capacidad que tienen los objetos de responder a la misma llamada o evento[1] dependiendo de los parámetros utilizados en la invocación, debido a la posibilidad de definir métodos o comportamientos con el mismo nombre en objectos de la misma clase.

- Programación estructurada: paradigma de programación cuyo objetivo es mejorar la calidad, claridad y el tiempo de desarrollo. La característica principal es que está formado por segmentos de código los cuales pueden estar compuestos desde una a varias instrucciones y cada uno de estos fragmentos debe tener un punto de entrada y de salida de los datos, lo que hace que sean leídos secuencialmente de principio a fin manteniendo la continuidad. Está compuesto por tres estructuras principales que son las siguientes:
  - Secuencia: es el orden el cual tienen que seguir las instrucciones al ser ejecutadas.
  - Condicional: es la posibilidad de elegir entre dos o más instrucciones en base a una condición.
  - Iteracción: es la ejecución de una instrucción repetidas veces mientras se cumple una condición.

- Es un lenguaje de alto nivel.
- Compilación: mediante el proceso de compilado es cual es realizado mediante un compilador, el código es traducido a bajo nivel para que pueda ser ejecutado en la máquina. Existe un gran número de compiladores para diferentes plataformas y sistemas operativos.

3.1.2. Python

Python es un lenguaje de programacion el cual destaca por su código fácil de entender y limpio. Algunas de las principales características de python son:

- Es un lenguaje interpretado, lo que significa que el código es traducido por medio de un intérprete cuando es necesario, al contrario de los lenguajes compilados como por ejemplo son java o C, los cuales son traducidos una única vez por un compilador mediante un proceso denominado compilado para ser ejecutado posteriormente por el sistema. De esta manera, nuestro código podrá ser ejecutado en cualquier sistema siempre y cuando este tenga un interprete de python. Al desarrollar scripts[2], que python sea un lenguaje interpretado es una ventaja ya que a la hora de desarrollar y realizar pruebas se mejoran los tiempos debido al ahorro de tiempo del proceso de compilado.
- Al igual que en 'C++', 'Python' es un lenguaje multiparadigma ya que soporta los siguientes paradigmas de programación:
  - Orientado a objetos : explicado en el apartado anterior de 'C++'.
  - Funcional: paradigma de la programacion declarativa que utiliza funciones matemáticas que permiten la variación del programa mediante la mutación de variables. Esto permitirá trabajar con variables de entrada y salida. Algunas de sus características principales son las funciones de orden superior las cuales se caracterizan por poder tomar otras funciones como parámetros o devolverlas como resultado. Otra característica importante

de la programación funcional es el uso de la recursividad, por lo que las funciones pueden invocarse a si misma tantas veces sea necesario hasta conseguir el objetivo o el caso final. El uso de la recursividad permitira desarrollar un código más corto y legible.

  - Imperativo: permite la modificación del estado del programa utilizando condiciones o instrucciones que le indican a la máquina o computadora cómo realizar la acción que modificará el estado.

- Tipado dinámico: En distintos momentos, una variable puede tomar valores los cuales pueden ser de distinto tipo. Al contrario que otros lenguajes de programación en python las variables son declaradas por el contenido que van a almacenar, no por el contenedor. Python detectará directamente el tipo la variable dependiendo del contenido de esta, por lo que no es necesario indicárselo.

- Licencia de código abierto

- Soporte para múltiple variedad de bases de datos

- Cuenta con una gran cantidad de librerías y recursos con especial énfasis en las matemáticas, lo cual es otra de las razones para ser el lenguaje utilizado en esta parte del proyecto.

'Pip' es el gestor de paquetes de python que va a ser utilizado en este proyecto para administrar e instalar los paquetes necesarios para el tratamiento de los datos y poder transformar esta información en gráficas para que los usuarios puedan interpretar la información de forma más sencilla. Al utilizar la versión de python 3.8, pip se encuentra integrado por defecto despues de la instalacion de Python.

PyCharm es un entorno de desarrollo para python el cual ha sido utilizado para el desarrollo de parte de este proyecto. Es un entorno multiplataforma el cual puede ser instalado tanto en Windows, Linux o Mac OS.

3.2. Frameworks

3.2.1. Robocomp

Robocomp es un framework libre y de desarrollo software para robots el cual sigue el paradigma de programación POC, programación orientada a componentes.

Este paradigma de programación se basa en que los usuarios construyen un mercado de componentes software para que las aplicaciones puedan reutilizar estos componentes que han sido creados anteriormente y se encuentran testeados. De esta manera, cuando se quiera construir una nueva aplicación, esta será creada de forma más rápida y robusta.

Además este paradigma de programación mejora algunos de los grandes problemas del desarrollo software, como son la encapsulación, seguridad y polimorfismo.

Un componente es una unidad de un programa software formado por interfaces y dependencias las cuales le permiten comunicarse con otros componentes.
Para que dos componentes se comuniquen mediante sus interfaces es necesario el uso de un middleware y en este proyecto el utilizado se llama Ice[3], ya que es el que utiliza Robocomp.

Como se ha explicado previamente, Robocomp es un framework de desarrollo software en el cual se pueden crear componentes de una manera sencilla e intuitiva. Para crear un componente se utiliza un archivo específico el cual tiene la extensión .cdsl[4]. En el capítulo cinco se explicará cómo es el archivo .cdsl de este proyecto.

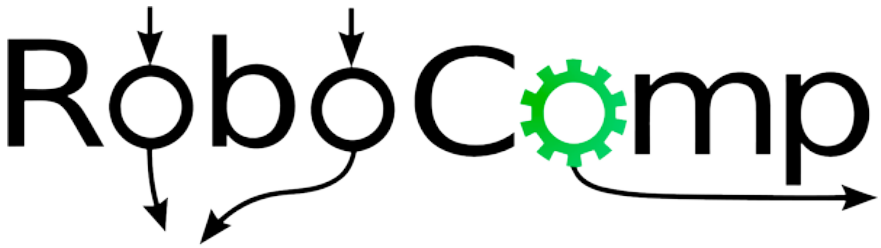


Figura 3.1: Robocomp

3.2.2. Robot Operating System

Robot Operating System, también conocido como ROS, es un framework para el desarollo software específico de robótica.

La filosofía de este framework es que todos los programas sean de libre uso, escalables e integrables a cualquier software de robótica, por ello sigue utilizando plataformas de software libre como KDL, OpenCV, etc.

La estructura de este framework es modular, ya que cada programa es denominado nodo, que a la vez se ejecuta dentro de un ejecutable que funciona como núcleo del sistema.

Figura 3.2: Robot Operating System

3.2.3. Qt

Qt es un framework de desarrollo de aplicaciones multiplataforma orientado a objetos que permite crear programas en linux, OS X, windows y android entre otros. Se creó como software libre y cuenta con una gran comunidad de programadores los cuales crean librerías de gran utilidad, que al ser de código libre pueden ser utilizadas por cualquier desarrollador. Es un framework desarrollado en C++.

Qt tiene su propio entorno de desarrollo integrado, llamado QtCreator. Este IDE no ha sido utilizado durante el desarrollo de este proyecto ya que para el desarrollo en C++

se utilizó el IDE Visual Studio Code.

Una ventaja muy importante que tiene este framework es que aunque esté escrito mayoritariamente en C++, existen bindings para algunos de los lenguajes más utilizados como son java y python, aunque estos no son desarrollados por Qt si no por terceros.

Este framework ha sido importante durante el desarrollo de este proyecto ya que se han utilizado algunas librerías como 'Qtime' y 'QPointF', las cuales han sido importantes para las principales funcionalidades de este proyecto.

Figura 3.3: Framework Qt

3.3. Simuladores

3.3.1. RCIS

El simulador RCIS es el simulador de robots de Robocomp, desarrollado por Robolab. Para ejecutar un mapa en este simulador será necesario un archivo de extensión .xml[5] con toda la información del mapa que queremos simular en su interior. Para ejecutar el simulador, será necesario escribir en la línea de comandos 'rcis' seguido del nombre del mapa que queremos abrir. En la siguiente imagen se verá un mapa simple con el simulador RCIS, sin ningún robot u obstáculo en él.

Figura 3.4: Mapa básico simulador RCIS

3.3.2. Gazebo

El simulador gazebo es un simulador 3D, dinámico y multi-robot, el cual permite evaluar el comportamiento de un robot en un mundo virtual, crear mundos virtuales usando herramientas sencillas, importar modelos ya creados y personalizar el diseño de los robots.

Gazebo es el simulador compatible con el framework ROS, nombrado anteriormente.

Figura 3.5: Gazebo

3.3.3. V-Rep

El simulador V-Rep es un simulador 3D, disponible para los principales sistemas operativos y de licencia libre. Está basado en una arquitectura de control distribuida, es decir, los scripts de control, pueden conectarse de forma directa a los objetos que existen en la escena y pueden ejecutarse de forma paralela.

Este simulador también tiene la ventaja de que te permite programar en muchos de los principales lenguajes como Python, C++, Java, Lua, etc. Además cuenta con una amplia documentación online de su uso para que sea más sencillo el aprendizaje de la herramienta.

Figura 3.6: V-Rep

3.4. Estructuras de control

3.4.1. Máquina de estados

Definición

Una máquina de estados, también conocido como autómata, es una abstracción matemática utilizada para diseñar algoritmos, compuesta por estados que hacen de intermediarios entre las entradas y las salidas, y transiciones por las cuales el objeto cambiará de un estado a otro diferente.

Elementos que lo forman

Las máquinas de estados están formadas por cuatro elementos principales que son las entradas, las salidas, los estados y las transiciones. Las cuales explicaremos a continuación:

- Entradas : señal[6] que le llega a un estado.
- Salidas : señal que genera un estado a partir del estado actual y de las señales de entradas que le han llegado.
- Estados : es la situación de un objeto durante la cual el objeto está realizando una acción o esperando que suceda algún evento.
- Transiciones : es la relación que existe entre dos estados. Cuando un estado realice alguna acción o ocurra algún evento y se cumplan algunas condiciones determinadas se ejecutará la transición y el objeto cambiará de estado.

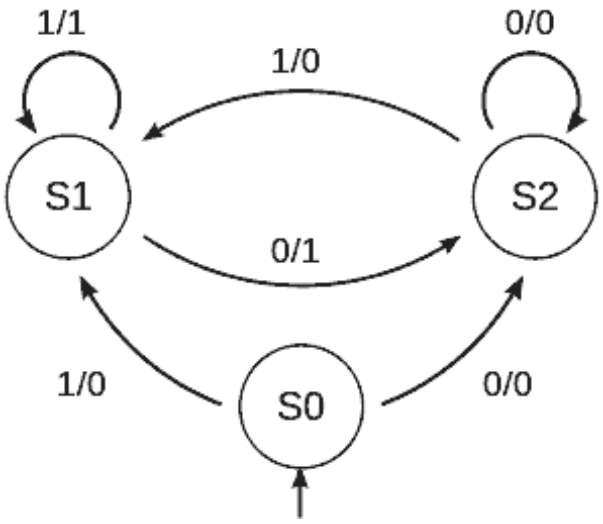


Figura 3.7: Ejemplo máquina de estados

Como podemos ver en la figura 3.7, en dicha máquina de estados existen tres estados diferentes S_0 , S_1 y S_2 , representados mediante círculos. Tambien podemos ver las transiciones representadas mediantes flechas. Además en cada transición de podemos ver dos números. El primero de ellos es la entrada y el segundo la salida.

Funcionamiento

La ejecución de los estados de la máquina van cambiando según las entradas que lleguen. Esto lo hacen mediante las transciones, como podemos ver en la figura 3.7. Una vez que una entrada llega y comienza a ejecutarse un estado, este devuelve una salida con todos los cambios implementados, ejecutando otra transición lo que hará que el objeto cambie a otro estado o vuelva al estado en el que se encuentra, a la espera de que suceda algún evento o llegue otra entrada.

3.4.2. Árboles de comportamiento

Durante el desarrollo del proyecto, se buscaron distintas tecnologías que pudiesen mejorar el funcionamiento de las máquinas de estados. Finalmente se decidió utilizar una tecnología muy usada en el mundo de los videojuegos y que poco a poco se va introduciendo en la robótica. Esta tecnología son los árboles de comportamiento.

Definición

Un árbol de comportamiento es una estructura de nodos jerárquicos que controlan la toma de decisiones de una entidad mediante inteligencia artificial.

Elementos que lo forman

Los árboles de comportamiento están formados por nodos y existen tres tipos principales de nodos, que son los siguientes:

- Compuesto: puede tener uno o más hijos. Estos hijos pueden ejecutarse en orden o de una manera aleatoria según se configure el nodo. Hay cuatro tipos de nodos compuestos que son los siguientes:
    - Secuencia: es un nodo el cual visita cada uno de los hijos en orden de izquierda a derecha. Si nada falla devuelve éxito al padre. Si cualquiera de los hijo falla, devuelve fallo. Funciona como una puerta lógica AND.

- Selector: es un nodo el cual ejecuta el primer hijo, si este falla pasa al siguiente hasta que alguno de los hijos devuelve éxito. En caso de no encontrar ningún hijo que retorne éxitosamente, devolverá fallo. Funciona como una puerta lógica OR.
- Secuencia aleatoria: funciona exactamente igual que la secuencia explicada anteriormente excepto que el orden que sigue es completamente aleatorio.
- Selector aleatorio:funciona exactamente igual que el selector explicado anteriormente excepto que el orden que sigue es completamente aleatorio.

- Decorator (simple): este nodo solo puede tener un hijo. Su función principal es cambiar el resultado que le devuelve el hijo. Hay cuatro tipos de nodos decorator, que son los siguientes:
  - Inversor: es un nodo que cambiara el resultado. Es decir, si es éxito lo cambiará a fallo y si es fallo lo cambiará a éxito.
  - Exitoso: este nodo siempre devolverá éxito.
  - Repetidor: este nodo hace que se repita hasta que devuelva un resultado.
  - Repetidor hasta que falle: este nodo hace que se repita hasta que devuelva fallo, y entonces el nodo decorator devolverá éxito a su padre.
- Hoja: son nodos que no tienen ningún nodo hijo. Son los más importantes ya que definen todas las tareas que realizará el árbol, pero a su vez son los de nivel más bajo en el árbol de comportamiento. Existen dos tipos nodos hojas diferentes y son las siguientes:
  - Acciones: son el nodos que ejecutan una acción.
  - Condiciones: son los nodos que comprueban una condición para ver si se puede ejecutar el siguiente nodo.

A continuación, se adjuntará una imagen en la cual se mostrarán algunos de los nodos explicados anteriormente y las iteraciones entre ellos.

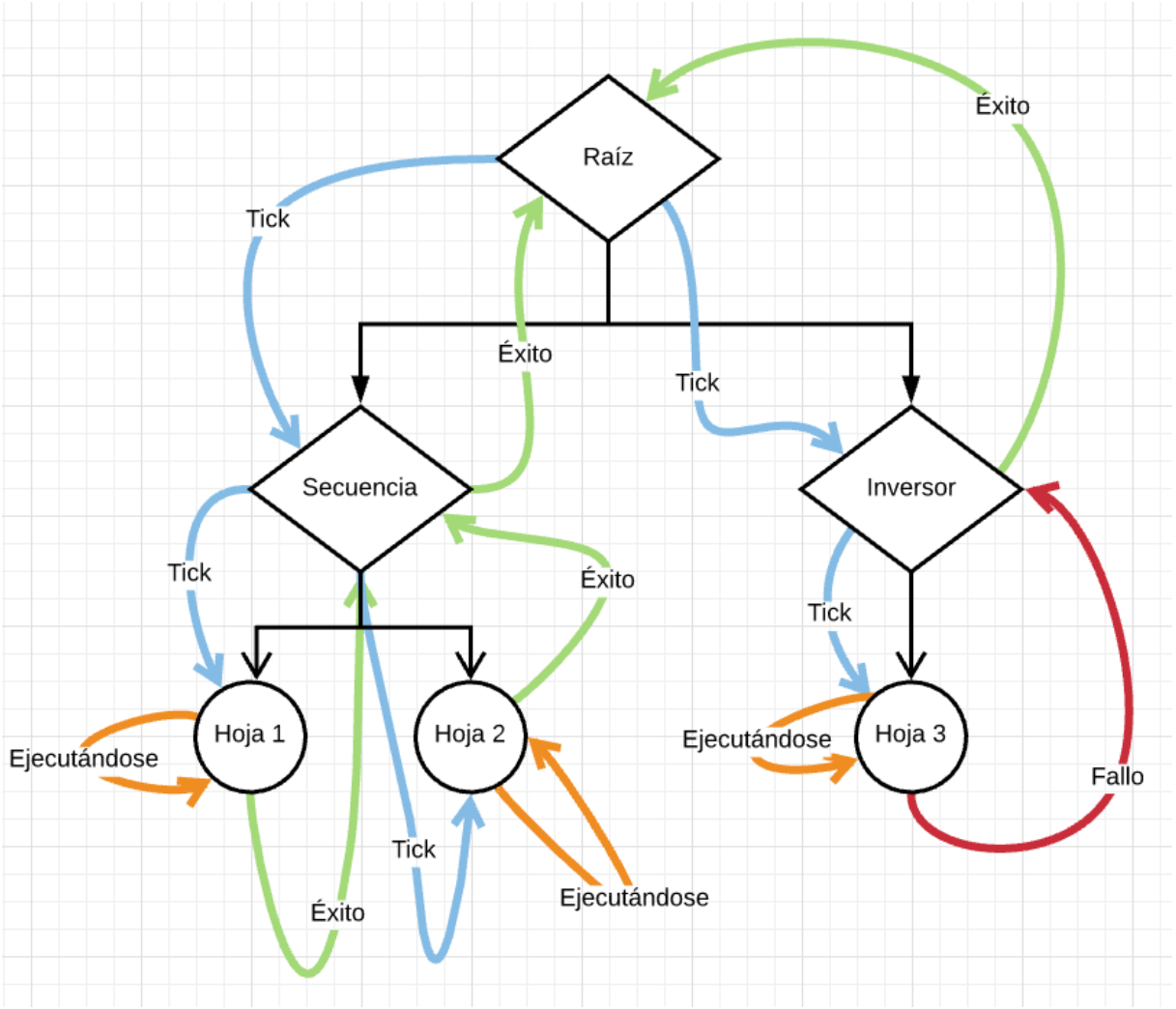


Figura 3.8: Ejemplo árbol de comportamiento

En la figura 3.8 se pueden observar los tres nodos principales que existen en los árboles de comportamiento. Tenemos dos nodos compuestos que son del tipo secuencias, estos son el nodo raíz y el nodo secuencia. También podemos ver tres nodos hoja y por último podemos ver un nodo decorator el cual es de tipo inversor.

Funcionamiento

El árbol de comportamiento funciona a traves de ticks[7], es decir, el root del árbol manda continuamente ticks sobre el nodo en ejecución. Los nodos se ejecutan y devuelven el estado en el que se encuentran. Existen tres estados diferentes, que son los siguientes:

- Éxito: todo ha ido bien y se puede pasar al siguiente nodo.
- Fallo: si algún nodo devuelve este estado, el árbol termina la ejecución.
- Ejecutando: este estado indica que el nodo todavía no ha terminado de ejecutarse.

El funcionamiento de los árboles de comportamiento se encuentra definido en la figura 3.8, ya que contiene todas las iteraciones posibles entre los tipos de nodos y con los distintos estados y transiciones posibles en cada caso.

En la imagen se puede ver como el primer tick que manda el nodo raíz, señalado del color azul es hacia el nodo secuencia. Este recibe el tick y vuelve a mandarlo hacia su primera hoja. La hoja uno se ejecuta, color naranja, y cuando termina de ejecutarse manda el tick como éxito, color verde, de nuevo hacia el nodo secuencia. Este nodo al recibir éxito de su nodo hoja pasa el tick al siguiente nodo hoja. El nodo hoja dos se ejecuta, si no hay problema devuelve éxito de nuevo a la secuencia. Como el nodo secuencia ya ha terminado de ejecutarse completamente manda el tick como éxito al nodo raíz. El nodo raíz al recibir éxito manda el tick al siguiente nodo, que en este caso es el nodo inversor. El nodo inversor manda el tick hacia la hoja, pero esta tras ejecutarse devuelve fallo por lo que manda el tick como fallo hacia el nodo inversor. Este nodo, cuya función es cambiar el resultado del nodo hijo, devolverá éxito a el nodo raíz y acabaría la ejecución del árbol de comportamiento. Hay que tener en cuenta que si un nodo que no sea de tipo hoja devuelve fallo hacia el nodo raíz la ejecución del árbol quedaría interrumpida.

Capítulo 4

Equipo utilizado

4.1. Hardware

En la parte final de proyecto, una vez la simulación de varios robots funcione correctamente, se utilizará una radio MDEK1001 con la cual se simulará el ruido que generan para introducirlo en el proyecto y conseguir así que el conjunto de datos obtenidos de la simulación sea más realista.

4.1.1. Radios MDEK1001

Las radios MDEK1001 poseen una placa base formada por antenas y sensores integrados. Estas radios se caracterizan porque soportan varios estándares de conectividad como son bluetooth y UWB (ultra wide-band).

Gracias a estas tecnologías, las radios tienen una gran precisión. Una de las ventajas de estas, es que pueden ser configuradas y monitorizadas mediante una aplicación móvil (solo válida para android superior a android 6.0) y un cliente web.

Estas radios son las que serán utilizadas durante los experimentos reales que se harán en los cebaderos. Por lo que se ha decidido que para hacer este trabajo más realista, se calculará el ruido que generan estas radios y se añadirá a los datos de posicionamiento

de los robots obtenidos en la simulación, para conseguir unos resultados lo más realistas posibles.

Figura 4.1: Rádio MDEK1001

Estos dispositivos cuentan con varios estándares de conectividad los cuales se explicarán a continuación. Para el desarollo del cálculo de los datos en animales reales, se hará uso de la tecnología UWB. Lo estándares de conectividad que soportan estas radios son:

- Bluetooth: es un protocolo de comunicación entre dispositivos de corto alcance, el cual se comenzó a desarrollar en 1994. Una de las grandes ventajas que posee esta tecnología es que se trata de una conexión automática que no requiere de ningún cable, es decir, los dispositivos se conectan de forma inalámbrica. Esta conexión inalámbrica se realiza a través de ondas de radiofrecuencias que se encuentran en la banda de los 2.4 Ghz. El ratio de funcionamiento del bluetooth es variable según la clase que tenga integrada nuestro dispositivo. Existen cuatro clases diferentes que son:

- Clase 1: los dispositivos conectados se deben encontrar en un ratio de hasta cien metros y tiene una potencia máxima de hasta 100 mW.
- Clase 2: su alcance de conectividad es de entre diez y veinte metros y tiene una potencia máxima de hasta 2.5 mW.
- Clase 3: su rango alcanza hasta un metro y tiene una potencia máxima de 1 mW.
- Clase 4: tiene un rango máximo de medio metro y una potencia máxima de 0.5 mW.

Un dispositivo bluetooth posee dos partes fundamentales para su correcto funcionamiento. La primera es un dispositivo de radio el cual se encargue de transmitir la señal, y una CPU[8] la cual se encargará de procesar estas señales.

- UWB: la banda ultraancha es una tecnología inalámbrica creada para la transmisión de datos digitales a corta distancia y a una densidad espectral[9] baja. Esta tecnología comenzó a desarrollarse sobre los años cincuenta pero estuvo limitado a aplicaciones militares hasta los años noventa. Hasta finales del siglo veinte no comenzó a utilizarse en dispositivos personales. Como su nombre indica, esta tecnología ocupa un gran ancho de banda en el cual usa señales de corta duración a través de un espectro que va desde los 3.1 GHz hasta los 10.6 GHz. Esto es una gran ventaja ya que UWB consigue transmitir más información que otras de estas tecnologías. Una de sus principales características es que funciona calculando el tiempo que tarda en volver la señal y tiene una gran penetración en distintos materiales. Esto hace que su precisión sea muy superior a la de otras tecnologías de comunicación.

La banda ultraancha tiene varias ventajes respecto al bluetooth. Las ventajas más importantes son las siguientes:

- Permite alcanzar velocidades de hasta 100 Mbps en 10 metros, 480 Mbps en un metro y hasta 1,5 Gbps lo que supone una velocidad mayor que la conseguida mediante bluetooth.

- Genera pocas interferencias de radio, y es muy eficiente energéticamente. Esto se debe a que la tecnología UWB transmite señales de muy baja potencia a través de un amplio espectro de frecuencia.
- La banda ultraancha ofrece una precisión que va desde los cinco a los diez centímetros mientras que la precisión que nos ofrece el bluetooth es de un metro.
- La tecnología de banda ultraancha es direccional, es decir, se sabrá la dirección la cual se encuentra la señal con una precisión de tres grados. Esta funcionalidad no ha sido añadida a bluetooth hasta su última versión.
- Tiene una gran capacidad para atravesar obstáculos. Las bajas frecuencias usadas en el espectro de frecuencia UWB tiene una gran longitud de onda, lo que hace que se utilice satisfactoriamente la comunicación Sin-Línea-de-Visión[10].

En la figura 4.2 añadida a continuación se puede ver la diferencia entre los principales estándares de comunicación entre los que se encuentran la UWB y el bluetooth.

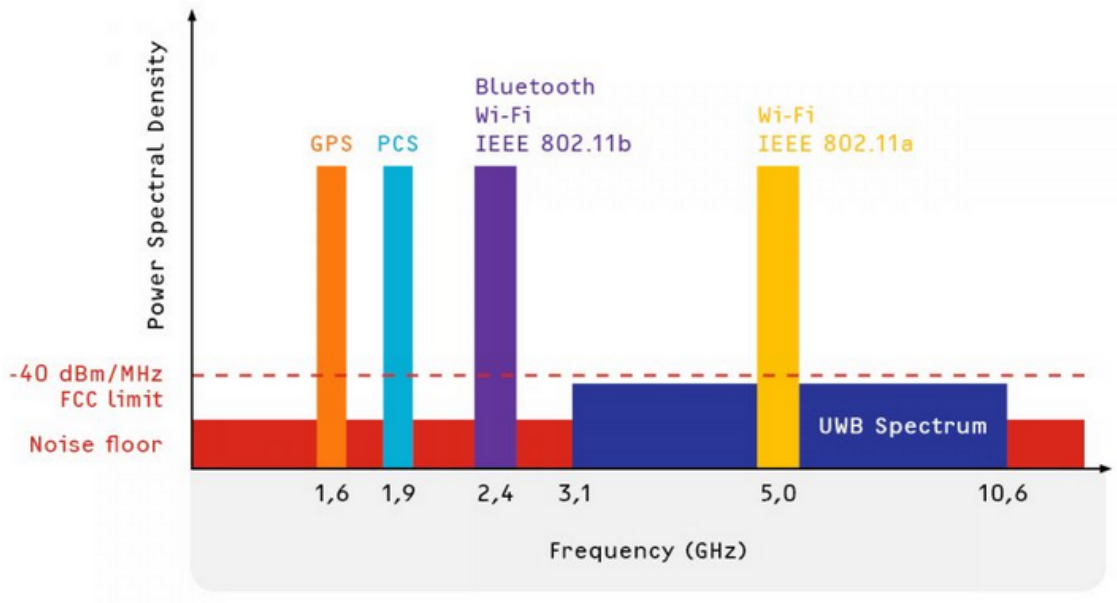


Figura 4.2: Comparación estándares de comunicación

En la figura 4.2 se puede apreciar como la banda ultraancha ocupa un gran espectro de frecuencia comparado a las demás tecnologías de comunicación. También se puede

distinguir que la densidad espectral utilizada por esta tecnología es muy inferior al resto de estas. Esto es una gran ventaja de la banda ultraancha sobre los demás estándares de comunicación ya que significa que la UWB consigue transmitir más información en un periodo más corto de tiempo.

Capítulo 5

Desarrollo del proyecto

En este capítulo se explicarán las distintas fases en las que se ha desarrollado el proyecto.

5.1. Problema a resolver

En este proyecto se han presentado dos problemas principales a resolver. El primero de ellos es la simulación de los comportamientos biológicos básicos de los corderos. Para ello, se crearán diferentes algoritmos mediante los cuales los robots virtuales simularán las acciones de comer, beber dormir y andar.

El segundo problema presentado es la localización de los robots en tiempo real. Con la obtención de estos datos se quiere realizar un análisis para estudiar el comportamiento de los animales dentro de los corrales de cebo.

Estos problemas han sido propuestos ya que uno de los objetivos del proyecto 'Caracterización etológica de corderos con tecnologías robóticas' es la localización de los corderos en tiempo real en los corrales de cebo. Para lograrlo hay que crear un modelo simulado mediante el cual se puedan realizar estudios e investigaciones ya que de esta manera se reduce el coste y el tiempo de estos estudios, en comparación a si estos estudios se hicieran en corrales de cebo reales.

5.2. Propuesta de solución al problema presentado

Ya conocidos los dos problemas a resolver en este proyecto, se procederá a explicar las soluciones propuestas para su resolución.

5.2.1. Simulación de robots

Para la simulación se usará el simulador RCIS ya nombrado anteriormente. El objetivo es conseguir desarrollar un software mediante el cual los robots virtuales puedan ejecutar las acciones biológicas básicas que realizan los corderos como son comer, beber, dormir y andar. Para ello se han utilizado las máquinas de estados y los árboles de comportamiento, cuya definición, configuración y funcionamiento se explicará en este capítulo.

5.2.2. Localización en tiempo real de robots

Para el desarrollo de esta fase, se sabe que durante la simulación se puede acceder directamente a las coordenadas exactas del robot en todo momento, pero esto no sería posible en un modelo en el entorno real, ya que la transmisión de los datos genera ruidos que producen como resultado que las coordenadas obtenidas no sean totalmente exactas, por tanto para hacer que la simulación sea lo más precisa posible a un experimento real, se calculará un margen de error aproximado del ruido que generaría una radio de las que serán utilizadas en el desarrollo del proyecto.

El cálculo del margen de error consistirá en recopilar datos reales producidos por una radio que ha sido colocada en una posición conocida en el laboratorio de Robolab durante una hora para comprobar los resultados obtenidos de las coordenadas de su posición, y mediante el cálculo de la desviación típica de estos valores, obtener el margen de error.

Teniendo en cuenta estos datos, se puede generar siempre un número aleatorio, dentro de los umbrales de error calculados previamente usando las radios, mediante una

distribución normal, para sumarle a nuestras coordenadas exactas y así generar dentro de la simulación el ruido generado por el funcionamiento de las radios.

Con esta solución se conseguirá que la simulación con robots virtuales sea un experimento similar sobre como funcionaría en robots físicos midiendo sus coordenadas con esta radios.

Con los datos obtenidos como se ha explicado, cada robot publicará las coordenadas en las que se encuentra en un archivo el cual se usará para generar unas gráficas desde donde se podrán analizar los datos de la simulación visualmente.

5.3. Iniciación del robot al entorno

Para crear el robot, el primer paso es instalar Robocomp en la máquina donde se realizará el trabajo. Una vez que RoboComp se ha instalado correctamente, el siguiente paso es crear un componente. Para realizar esta acción, el primer paso es crear una carpeta y dentro de esta ejecutar el siguiente comando:

```
robocompdsl cordero.cdsl
```

Este comando generará el archivo .cdsl que contrendrá la configuración de las comunicaciones del robot. Este archivo se tendrá que modificar para que el robot funcione correctamente.

Para representar el robot en el simulador hay que crear un archivo .xml con la configuración sobre la posición inicial del robot y del láser en el mapa. El ejemplo de la configuración para el primer robot creado en el proyecto es la siguiente:

```
<differentialrobot id="base1" port="11004">
<mesh id="base_robex1"
file="/home/robocomp/robocomp/files/osgModels/robex/robex.ive"
    tx="0" ty="0" tz="-0" scale="1000" collide="1"/>
```

```
        <translation id="laserPose1" tx="0" ty="140" tz="200">
                <laser id="laser1" port="11003" measures="1000"
                min="100" max="30000" angle="3" ifconfig="10000" />
                <plane id="sensorL1" nz="1" pz="-200" size="100"
                repeat="1" texture="#ff0000" />
    </translation>
<translation id="robotGeometricCenter1" tx="0" ty="0" tz="0">
</translation>
</differentialrobot>
```

Las primeras líneas seleccionan la imagen que se mostrará en el simulador para el robot, así como su posición en el mapa junto a la del laser.

A continuación se adjunta una imagen en la cual se puede observar el robot con el láser en el mapa.

Figura 5.1: Ejemplo robot con láser

Una vez que el archivo cordero.cdsl está correctamente creado y configurado, se ejecutará el siguiente comando para que se creen todos los archivos necesarios para poder lanzar el robot.

```
robocompdsl cordero.cdsl .
```

Este comando creará todos los archivos C++ en este caso, los cuales se modificarán para que el robot se comporte de la manera más parecida posible a un animal real.

5.4. Creación y desarollo de la máquina de estados

Para que el robot siga un comportamiento lo más similar posible a un cordero real, se ha creado una máquina de estados en la que se simulan los principales comportamientos básicos de los corderos.

La máquina de estados está compuesta por los estados por los que pasará el robot durante la ejecución y por las interacciones entre ellos, ya que un estado puede pasar a uno u otro distinto dependiendo del valor de la variable que controla la máquina de estados denominada 'Estado'. Esta variable contendrá el valor del nombre del estado en el que se encuentre en cada momento el robot y se irá actualizando una vez se realicen las acciones definidas para cada estado y siempre dependiendo del resultado de esa acción.

A su vez, la máquina de estados trabaja con otra variable denominada 'EstadoEnUso' la cual almacena la acción que se encuentra ejecutando el robot.

La diferencia entre las variables 'Estado' y 'EstadoEnUso' es que la primera puede almacenar todos los estados que se explicarán en el siguiente punto, mientras que la segunda únicamente podra almacenar las acciones que realiza el animal (comer, beber, dormir y andar) y que se intentan simular mediante los robots.

A continuación se adjunta una imagen en la cual se mostrará la estructura de la máquina de estados definida inicialmente para este proyecto, en la que se incluyen todos los estados con los que trabajarán los robots así como las posibles combinaciones que pueden producirse dependiendo de las entradas y salidas que se produzcan durante la ejecución de las acción definidas por cada estado. Como se puede ver en la figura 5.2, ciertos estados pueden ser procesados despues de validar las salidas de estados anteriores. Esto es debido a que ejecutan acciones intermedias como son colocarse o ir hacia cierto objetivo cuya finalidad es realizar una acción final de las que se desea simular en este proyecto como son comer o dormir.

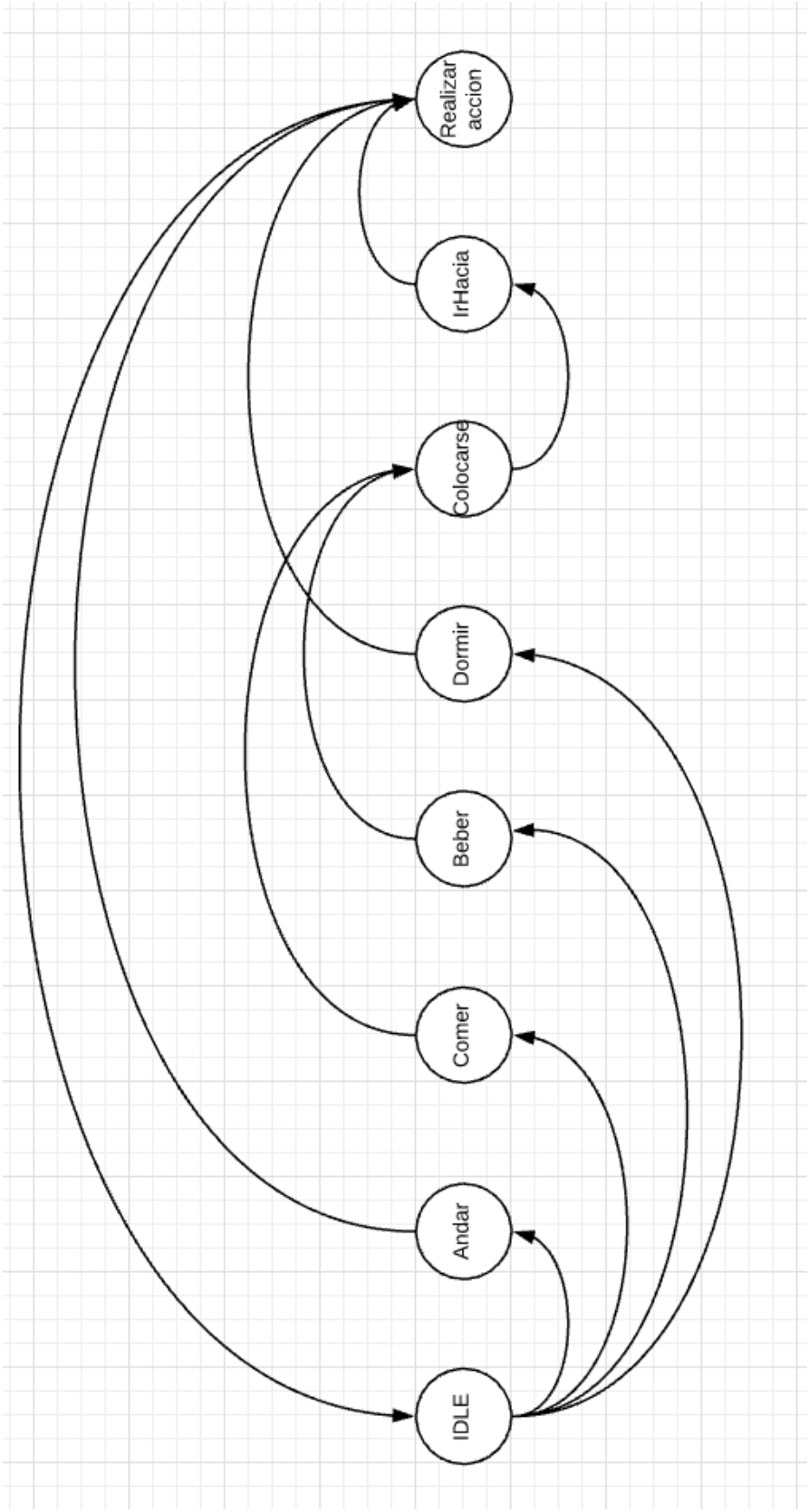


Figura 5.2: Máquina de estados.

5.5. Funcionamiento de la máquina de estados

Como se puede observar en la imagen anterior, la máquina de estados consta de ocho estados diferentes. A continuación se analizará el funcionamiento de cada uno de ellos.

- IDLE: el estado IDLE es el estado inicial de la máquina de estados. Este estado se encarga de elegir cual será la siguiente acción que realizará el robot. Esto se realizará de manera aleatoria sin poder repetir un mismo movimiento dos veces seguidas.

- Andar: el estado andar es el que se encarga de hacer que se mueva el robot por el mapa. Lo primero será controlar el tiempo que el robot se está moviendo de forma aleatoria por el mapa. Este estado, tiene tres opciones internas, que son las siguientes:

    - Sin obstáculos: en este caso el robot comprueba mediante su propio láser comprueba que no tiene ningún objeto delante y puede avanzar sin problema.

    - Caso objeto: en este caso el robot detecta que tiene muy cerca un objeto y antes de cruzarse con él, comprueba por qué lado esquivaría antes el objeto girando hacia izquierda o derecha y después sigue desplazándose por el mapa. Este mismo caso, es el que usa el robot cuando se encuentra con una pared. Además, este caso también será utilizado cuando el robot se cruce con otro que se encuentre en el mapa, ya que lo tratará como si fuera un objeto.

    - Caso esquina: en este caso el robot detecta que se ha quedado bloqueado en una esquina y no tiene una salida para ninguno de los dos lados, por tanto cuando detecta que está bloqueado, se queda parado y comienza a girar sobre su propio eje hasta encontrar la primera salida posible. Una vez la ha encontrado, sale de la esquina y vuelve a caminar por el mapa.

A continuación se muestran varias imágenes, las cuales se han obtenido durante las pruebas de los métodos explicados anteriormente, en las que se pueden observar los distintos casos que se puede encontrar el robot en el mapa durante el estado andar.

(a) Caso esquina (b) Caso pared (c) Caso objeto

Figura 5.3: Estado andar

- Comer: este estado se divide en dos acciones. La primera es cambiar el valor de la variable 'Estado' a colocarse. La segunda es cambiar el valor de la variable 'EstadoEnUso' a comer, que será utilizada más adelante por otros estados.
- Beber: este estado se comporta igual que el estado comer. En el caso de la variable 'Estado' también lo cambia a colocarse pero en el caso de la variable de 'EstadoEnUso' lo pone al valor beber.
- Dormir: en este estado se realizan dos acciones como en los dos estados anteriormente explicados. El primer caso es cambiar el valor de la variable 'Estado' a realizarAccion. El segundo caso es cambiar el valor de la variable 'EstadoEnUso' a dormir.
- Colocarse: este estado lo primero que hace es comprobar cual es el valor de la variable 'EstadoEnUso'. Después crea una variable del tipo QPointF en la cual, si el 'EstadoEnUso' es comer guarda las coordenadas donde se encuentra el comedero y si la variable 'EstadoEnUso' es beber, guardará las coordenadas

del bebedero. Una vez que ya tiene las coordenadas correctamente guardadas, el robot comienza a girar hacia al punto al que quiere dirigirse. Cuando el ángulo entre el robot y el punto al que quiere dirigirse es menor de 0.001 radianes, el robot se para y cambia el valor de la variable 'Estado' a IrHacia.

- IrHacia: en este estado, lo primero que hace el robot es comprobar cual es el valor de la variable 'EstadoEnUso', para guardar las variables del comedero o bebedero dependiendo donde vaya a desplazarse el robot. Tras esto, este estado pone al robot en movimiento, avanzando en línea recta, y va comprobando mediante las coordenadas guardadas del objetivo donde queremos llegar y una vez que hemos llegado, para al robot, y cambia el valor de la variable 'Estado' a realizarAccion. En este punto hay que destacar que si durante este estado, el robot se encuentra con algún obtaculo, es este caso otro robot que pueda estar ejecutándose, este lo esquivará y recalculará de nuevo la posición del comedero ó bebedero para continuar avanzando hasta su posición.

- Realizar acción: en este estado, utiliza la variable 'EstadoEnUso' para decidir cual de todas las acciones tiene que llevar a cabo. Esto lo hará mediante un switch/case, y en el que dentro de cada uno de los casos cargará el tiempo que tiene que estar realizando dicha acción y el mensaje que mostrará tras la ejecución. Tras ello, quedará esperando en un bucle el tiempo necesario para realizar la acción. Una vez que ha terminado, mostrará un mensaje por pantalla que mostrará la acción que ha llevado a cabo y el tiempo que ha estado realizando la acción. Para finalizar, cambia el valor de la variable 'estado' a IDLE y así el robot comenzará a realizar una nueva ejecución.

En la figura 5.4 se muestra el diagrama de flujo de la máquina de estados desarrollada para este proyecto. Los estados y las acciones se muestran mediante rectángulos, las condiciones mediante rombos y las iteracciones entre ambos mediante flechas.

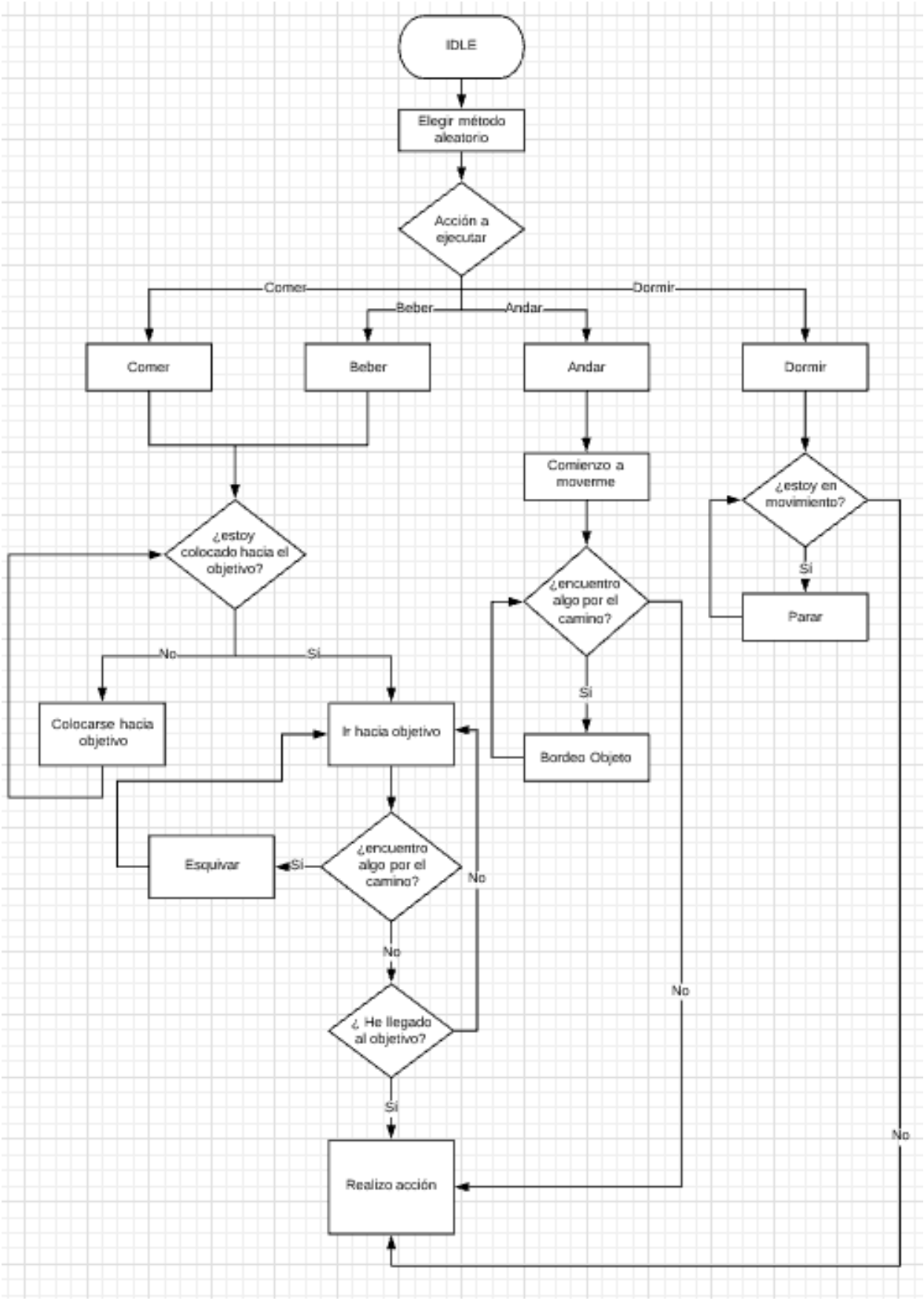


Figura 5.4: Diagrama de flujo de la máquina de estados.

5.6. Creación y desarrollo del árbol de comportamiento

Una vez que el robot se encuentra funcionando correctamente utilizando la máquina de estados, se pasará a implementar dicho robot utilizando un árbol de comportamiento ya que se quiere comprobar el funcionamiento de esta estructura aplicado a la robótica.

La librería que se ha utilizado para la creación y desarrollo de este árbol de comportamiento se llama 'BrainTree'. El enlace a su documentación se encuentra en el apartado de la bibliografía.

En la siguiente imagen se muestra la definición del árbol de comportamiento utilizado en este trabajo. La definición, creación e implementación del árbol de comportamiento realizado en este proyecto se encuentra en el enlace añadido en la bibliografía de git.

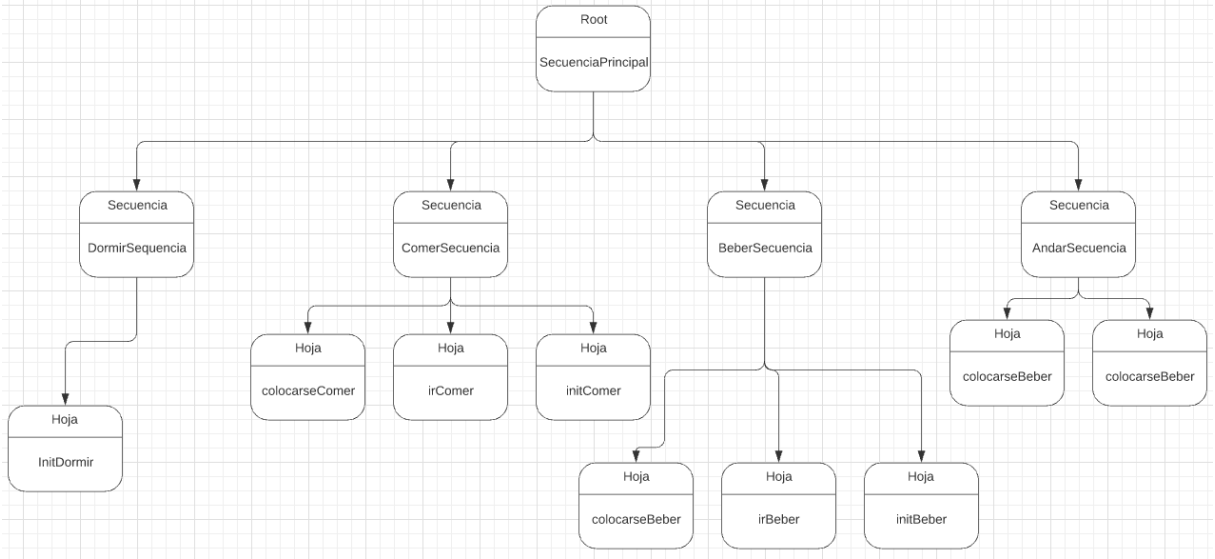


Figura 5.5: Árbol de comportamiento.

5.7. Funcionamiento del árbol de comportamiento

Como se ha explicado anteriormente, el árbol funciona mediante ticks, los cuales son mandados a las secuencias y a su vez a la hoja[11] correspondiente para que esta se ejecuta. Cuando el nodo devuelve éxito pasa al siguiente. Seguidamente, se explicará como funciona cada una de las partes del árbol.

5.7.1. Secuencia principal

La secuencia principal es la que se encarga de mandar los ticks a las demás secuencias. Esta tiene como hijos las otras cuatro secuencias principales, que son las que se encargan de realizar las cuatro acciones que realiza el robot simulando un animal real. Se encarga de ejecutar en orden las secuencias de dormir, comer, beber y andar. Una vez que termina una ejecución, vuelve a comenzar y así el robot nunca se encuentra sin realizar ninguna acción.

5.7.2. Secuencia dormir

Esta secuencia es la que se encarga de realizar la acción de dormir y tiene un nodo hoja, que es el que ejecuta la acción.

- initDormir: esta hoja se encarga de simular el comportamiento de dormir. Para llevarlo a cabo, para el robot, y se este queda en la misma posición durante un rango de tiempo que va desde los cinco a los diez segundos. Durante este tiempo, todos los ticks que le llegan los devuelve con el valor 'ejecutando'. Una vez pasa ese tiempo, el siguiente tick que le llegue le devolverá como éxito y así el siguiente tick hará que se ejecute el siguiente nodo.

5.7.3. Secuencia comer

Esta secuencia es la que se encarga de realizar la acción de comer. Para realizar esta acción completa se requiere de diferentes acciones , por lo tanto esta secuencia la forman tres hojas, que son las siguientes.

- colocarseComer: esta hoja es la encargada de que el robot se posicione de frente al comedero. Esto lo hace parándose y girando sobre su eje hasta que el ángulo que forma con el comedero es menor de 0.001 radianes. Durante todo este tiempo que está calculando el ángulo, cada tick que le llega le devuelve con el valor de ejecutando, y hasta que no comprueba que el ángulo es menor al valor comentado anteriormente, no manda el tick como éxito para que pueda ejecutarse el siguiente nodo.

- irComer: esta hoja se encarga de poner en movimiento al robot, una vez que ya está colocado, hacia el comedero. En el momento que empieza a moverse todos los ticks los devuelve como ejecutando. Cuando comprueba que ha llegado al comedero, para el robot y manda el siguiente tick como éxito. Si por el camino hacia el comedero, el robot se cruza con otro robot, este le esquivará y así evitarán una colisión.

- initComer: esta hoja se encarga de realizar la simulación de la acción comer. Una vez el robot está en el comedero, este se estará quieto durante un rango de entre dos y seis segundos. Cuando termine este tiempo, el siguiente tick lo devolverá como éxito y el nodo de la secuencia principal podrá ejecutar la siguiente secuencia, ya que la accion 'comer' ha terminado correctamente.

5.7.4. Secuencia beber

Esta secuencia es la que se encarga de realizar la acción beber. Al igual que la secuencia de comer, esta secuencia la forman varias hojas ya que el robot para beber tiene que desplazarse hacia el bebedero y no siempre está ni cerca, ni centrado respecto al bebedero. Estas tres hojas son las que forman esta secuencia:

- colocarseBeber: esta hoja es la encargada de que el robot gire sobre su eje y quede colocado enfrente del bebedero. Para ello, estando parado, el robot gira sobre su eje y cuando el ángulo que forma el robot respecto al bebedero es

menor que 0.001 radianes, en el siguiente tick devolverá el valor éxito y pasará a ejecutarse el siguiente nodo.

- irBeber: una vez que el robot está colocado, esta hoja se encarga de que el robot avance hacia el bebedero. Mientras avanza hacia este, va comprobando si ya ha llegado mediante las coordenadas, y todos los ticks los devuelve como ejecutando. El momento que el robot detecta que ha llegado al bebedero, el robot se para y devuelve el siguiente tick que le llega como éxito. Si por el camino hacia el bebedero, el robot se cruza con otro , este le esquivará utilizando el mismo proceso que en el caso de la acción 'comer'.

- initBeber: esta hoja se encarga de realizar la simulación de la acción beber. Para ello, queda parado el robot durante cinco segundos. Una vez que pasa ese tiempo, termina y manda el tick con el valor de éxito y así el árbol puede pasar a la siguiente secuencia ya que la secuencia beber ha terminado satisfactoriamente.

5.7.5. Secuencia andar

Esta secuencia es la que se encarga de que el robot se mueva por el mapa siguiendo una ruta aleatoria durante un periodo de tiempo determinado, sin tener una posición final determinada. Para realizar esta secuencia completa, tienen que ejecutarse correctamente dos hojas. Estas hojas son las siguientes.

- initAndar: esta hoja es la encargada de iniciar el contador de tiempo que el robot tiene que estar caminando por el mapa. Una vez que resetea esa variable, el siguiente tick que le llega le devolverá como éxito.

- actionAndar: esta hoja es la encargada de poner en movimiento al robot por el mapa. Para ello controla el tiempo que lleva moviéndose y cuando llega a los quince segundos , el siguiente tick que reciba será devuelto como éxito. Además, mediante el láser del robot, controla que en todo momento no choque con ningún objeto, ni con ningún otro robot. Cualquier objeto que se encuentro lo rodeará y seguirá su camino.

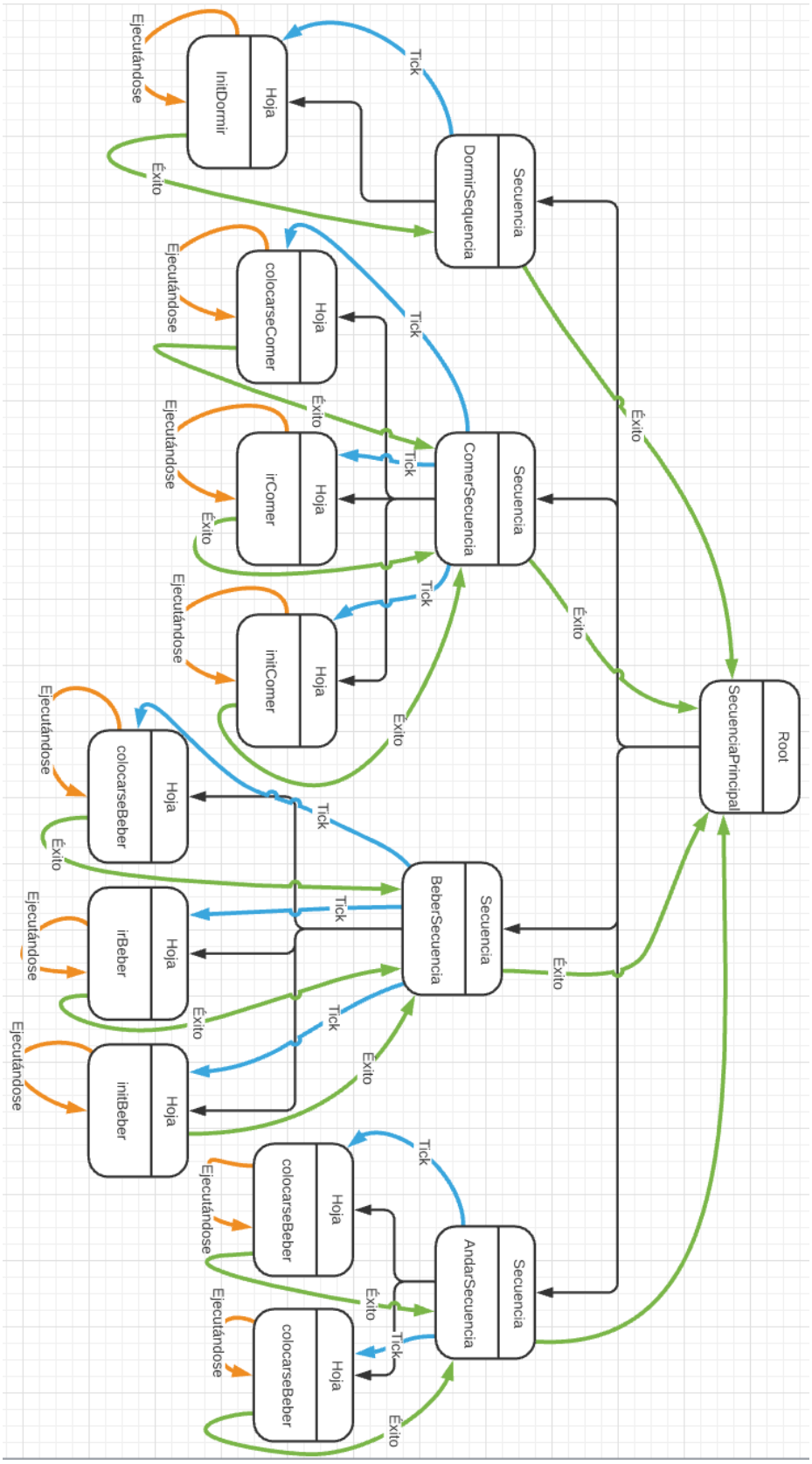


Figura 5.6: Diagrama de funcionamiento del árbol de comportamiento.

En la figura 5.6 se puede observar el como es el funcionamiento completo del árbol de comportamientos. En color azul tenemos representados los ticks que manda el nodo raíz hacia las secuencias y los ticks que mandan las secuencias a los nodos hoja. En color naranja vemos representada la ejecución del nodo hoja. Finalmente en verde vemos como los nodos hojas devuelven el valor éxito en los ticks a la secuencia y cuando finalmente le llega el tick de éxito del último nodo hoja, esta lo devuelve hacia el nodo raíz para que este pueda enviar el siguiente tick a la secuencia que le corresponde.

5.8. Introducción final de cinco robots en el entorno

Una vez que un robot se encuentra funcionando correctamente mediante el árbol de comportamiento, el siguiente paso es incorporar un nuevo robot para comprobar los posibles problemas que pueden aparecer cuando hay más de una instancia de robot ejecutándose en el mismo mapa . Se decide comenzar esta prueba únicamente con dos robot porque es la manera más simple de identificar los errores que pueden aparecer. Una vez que estos errores se localicen y se solucionen, se procederá a añadir todos los necesarios para llevar a cabo la finalidad de este proyecto. Los errores que se han encontrado son los siguientes:

- Choques cuando los robots se encontraban en la función andar: estudiando el comportamiento de los robot, se observó que no se había tenido en cuenta la posibilidad de que dos robots chocasen entre ellos.
  Para solucionar este error, se implementó una función en la cual si un robot detectaba algún objeto en su láser y estaba dentro de un pequeño umbral, el robot giraría y tomaría otra dirección.

- Error en las funciones en las cuales se llamaba al robot mediante su id: este error se encontró cuando se observó que el segundo robot actualizaba las coordenadas con las del primero ya que tenían el mismo identificador en el archivo de configuración de los robots.

Para solucionar este problema, se creó una variable dentro del fichero de configuraciónes en la que se indica el id/nombre del robot y será esa variable la utilizada por las distintas funciones para que los robots sean independientes unos de otros y no compartan variables ni estados.

- Choque cuando se encontraban en las funciones IrComer e IrBeber: en este caso nos encontramos con el mismo problema que se ha comentado en la función andar, pero este problema hay que resolverlo de problema diferente ya que el robot va camino al comedero/bebedero entonces no puede girar simplemente en otra dirección sino que, una vez ha esquivado el obstáculo, debe recalcular la posición del comedero ó bebedero para terminar de realizar la acción en la que se encontraba.

  Para ello se ha implementado una función en la cual se define que si el robot se encuentra con un obstáculo, este lo rodea por la izquierda y vuelva a colocarse en dirección al comedero/bebedero. Esta solución es válida se encuentre a uno o más robots por el camino.

- Error del tiempo realizando una acción: al estudiar la ejecución de los cinco robots, se comprobó que el robot se quedaba realizando una acción durante un tiempo fijo, algo que no concuerda con el comportamiento de un animal real.
  Para ello, se sustituyó el número fijo de segundos que el robot tenía que estar realizando una acción, por un número aleatorio con una distribución normal.

- Los robots realizaban las acciones en el mismo orden debido a que utilizaban el mismo árbol, por lo que se perdía bastante tiempo al comienzo de la ejecución porque todos los robots intentaban esquivar a los demás. Se decidió generar varios árboles con distinto orden de las secuencias para conseguir de esta manera visualizar mejor a los robots durante su ejecución y evitar problemas de rendimiento al comienzo de esta.

A continuacion se mostrará una imagen en la que se puede observar a varios de los robots en el mapa durante la misma ejecución.

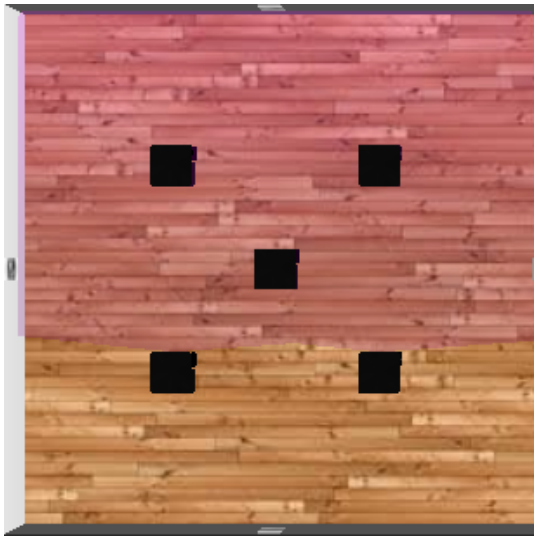


Figura 5.7: Múltiples robots en el mapa.

5.9. Publicación de las coordenadas

La última función que se ha implementado en el proyecto ha sido la publicación de la posición de los robots en tiempo real. En la simulación, en cualquier momento es posible acceder a la posición exacta en el mapa del robot y eso no es del todo correcto debido a que las radios emiten las coordenadas con un margen de error, ya que generan un ruido que hace que no sean totalmente exactas.

Por lo tanto, se decidió calcular un margen de error aproximado utilizando una radio MDEK1001 explicada anteriormente, la cual estuvo situada durante un tiempo en una localización específica de la cual se conocían las coordenadas exactas, por lo tanto durante ese tiempo estaba emitiendo las coordenadas con un margen de error, el cual se obtuvo para ser utilizado en la simulación.

Una vez se tienen esos datos, se genera un número aleatorio dentro del rango de error obtenido, a través de una distribución normal, y se le sumará a las coordenadas que

publicarán los robots.

Estas coordenadas serán almacenadas en ficheros de texto con el margen de error ya aplicado, uno por cada robot existente y estos ficheros serán utilizados posteriormente para la generación de gráficas que facilirán la compresión de estos datos. Las coordenadas se almacenarán el los ficheros siguiendo el formato 'x#y' siendo 'x' la coordenada del robot respecto al eje de abscisas del mapa y siendo 'y' la coordenada del robot respecto al eje de ordenadas del mapa.

5.10. Generación de gráficas a partir de los datos obtenidos

Una vez se han obtenido en los ficheros las coordenadas de los robots, vamos a utilizar esa información para generar gráficas que puedan ayudar a la compresión de estos datos visualmente.

Para realizar el desarrollo de estas gráficas, vamos a utilizar python como lenguaje de programación, para crear varios scripts los cuales contendrán la creación de distintas gráficas como la ruta de los robots, tanto grupalmente como individualmente o mapas de calor para identificar las zonas en las que los animales pasan más tiempo, y de esta manera poder optimizar el posicionamiento de los comederos y bebederos a zonas más cercanas a las posiciones más concurridas o realizar otro tipo de mejoras que ayuden a mejorar el proceso.

Aunque se van a desarrollar scripts, para facilitar el desarrollo se va a utilizar el entorno de desarrollo 'PyCharm'.

Para instalar python en el equipo simplemente se debe ejecutar el comando 'sudo apt-get install python3' y se instalará la última versión disponible, en este caso, la versión 3.8. Una vez instalado python y descargado el entorno de desarrollo, se

procederá a descargar los paquetes necesarios que contendran la funcionalidad para generar las graficas. Estás librerias son:

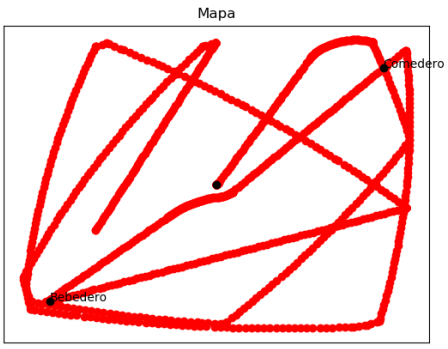
- numpy: contiene funciones matemáticas de alto nivel para trabajar con matrices y, como en este caso, vectores, los cuales almacenarán las coordenadas una vez estas hayan sido procesadas.
- matplotlib: es una librería que permite la generación de distintos tipos de gráficas como diagramas de barras, diagramas de dispersión o histogramas utilizando estructuras de datos como vectores o matrices.

Para descargar estos paquetes en la máquina de trabajo, será necesario lanzar el comando 'python pip install 'packageName".

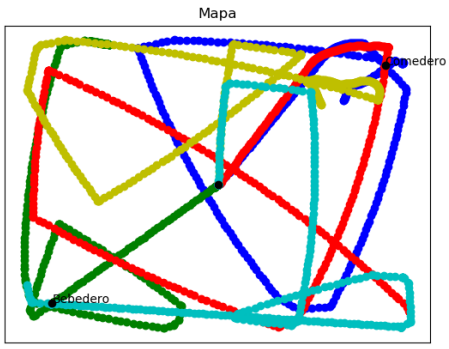
El trabajo realizado en los scripts se centra principalmente en la lectura de los ficheros generados durante la ejecución de la simulación de los robots. Los datos son escritos en un formato del tipo x#y en los ficheros, por lo que hay que extraer ambas coordenadas, las cuales serán almacenadas en distintos vectores que será utilizados posteriormente como datos de entrada para generar las gráficas.

El resultado del desarrollo de los scripts son la graficas que se verán a continuación las cuáles contienen:

- Mapa del recorrido de uno y varios robots por el mapa.



(a) Recorrido robot individual



(b) Recorrido grupo de robots

En las gráficas mostradas anteriormente se indican mediante puntos la posición inicial de los robots, coordenadas (0,0), y las posiciones del comedero y del bebedero. Se puede comprobar en ambas imágenes como los robots han pasado tanto por el comedero y el bebedero para luego seguir su recorrido por todo el mapa.

En la figura (b) se puede comprobar como algunos robots realizan alguna variación en su recorrido rectilíneo debido a que por el camino se ha cruzado con otro robot y ha tenido que esquivarlo y recalcular su posición.

- Mapa de calor de las zonas mas transistadas por lo robot, dividido en cuadrantes.

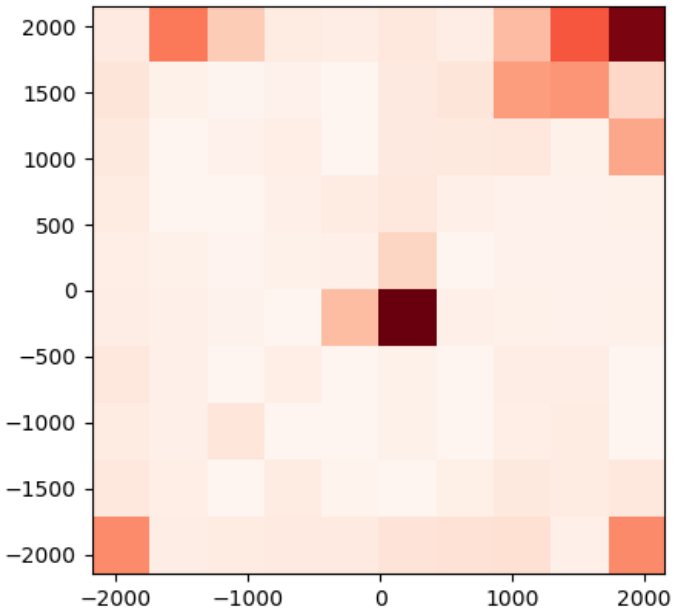


Figura 5.8: Mapa de calor.

Como se puede observar en la figura 5.8, las zonas más concurridas, representadas mediante colores cálidos, se corresponden la zona central desde la cual comienza la ejecución de los robots, las zonas cercanas al comedero y las zonas cercanas al bebedero. También se destacan algunas zonas cercanas a las paredes ya que cuando

un robot va a chocar, se para para comenzar a girar, entonces pasa más tiempo en esa posición.

Este gráfico ha sido tomado con las coordenadas de los ficheros generados por los robots durante la simulación simultánea de varios robots en el mismo mapa.

Capítulo 6

Resultados y líneas futuras

Una vez que el proyecto se ha desarrollado al completo, hay que comprobar si los objetivos impuestos inicialmente se han cumplido y cuales han sido los resultados.

6.1. Resultados

Inicialmente se estableció como objetivo principal conseguir que cinco robots recrearan el comportamiento fisiológico de un cordero y que cada uno publicara su localización en tiempo real cada cierto umbral de tiempo en un fichero.

Para ello, existían varias tareas principales a desarrollar que eran las siguientes:

- Implementar las funciones básicas que realiza un cordero en un robot mediante máquinas de estado.
- Estudiar funcionamiento y creación de árboles de comportamiento. Crear el mismo sistema implementado mediante máquinas de estado pero con árboles de comportamiento.
- Introducir un segundo robot en la simulación y estudiar qué errores pueden aparecer y la iteracción que existe entre ambos robots.
- Introducir los cinco robots.

- Publicación de la posición de los robots en tiempo real.

En general, todos estos objetivos que fueron marcados han sido desarrollados y probados para certificar su correcto funcionamiento. Cabe decir, que se trata de una versión inicial la cual se actualizará y ampliará a medida que se desarrollen nuevas funcionalidades.

6.2. Líneas futuras

Como hemos mencionado anteriormente, el sistema que ha sido desarrollado en este proyecto se trata de una versión inicial. Esta versión puede ser mejorada y ampliada. Algunas de estas ideas de ampliación surgieron durante el proceso del desarrollo de este y ayudarán a mejorar el rendimiento y los resultados de este proyecto en el futuro. Algunas de estas mejoras son:

- Modificación de la librería utilizada de los árboles de comportamiento: con un problema que nos encontramos a la hora de utilizar el árbol fue que al cruzarse dos robots en las acciones de irComer y irBeber, no podíamos volver al nodo padre(secuencia) para volver a realizar la acción de colocarnos hacia nuestro objetivo.

  Esto se debe a que mediante los tres valores con lo que se devuelven los ticks solo se puede avanzar hacia el siguiente nodo, quedarnos ejecutando el nodo en el que nos encontramos o bien parar la ejecución del árbol.

  Por lo que una opción que se pensó fue modificar la librería del árbol de comportamiento y crear un nuevo valor para devolver en los ticks, el cual pueda volver la ejecución al nodo padre y así poder comenzar de nuevo la secuencia.

- Añadir más funcionalidades secundarias los robots y así hacer que el modelo sea más real. Algunas de estas funcionalidades pueden ser por ejemplo jugar, en la que dos o más robots interactúen entre ellos.

- Crear variables que simulen si el animal tiene hambre o sed, las cuales se carguen cuando se esté en el comedero y el bebedero pero se vayan descargando mientras el robot va realizando otras acciones por el mapa y así el robot vaya a comer/beber solo cuando esta variable tenga un nivel bajo y necesite recargarla. De esta manera se podrían gestionar los ticks mediante algunos eventos.

- Implementar una interfaz gráfica desde la cual se puedan exportar los ficheros en los cuales los robots publican sus coordenadas y generar las gráficas a partir de cada fichero.

- Implementar este sistema en robots reales a través de LearnBlock[12]. De esta manera se daría un paso más hacia la implementación en animales reales.

Capítulo 7

Conclusiones

El proyecto en el que se engloba este trabajo se llama 'Caracterizacion etológica de corderos con tecnologías roboticas', en el cual se encuentra trabajando actualmente Robolab. Como se ha comentado en el apartado anterior, el resultado del proyecto parece satisfactorio ya que se ha conseguido emular las funciones básicas de un animal como son comer, beber, andar y dormir mediante robots virtuales, utilizando las herramientas explicadas en este documento, que era uno de los objetivos principales que se propuso al inicio del proyecto, junto al posicionamiento en tiempo real del animal, que también se ha simulado utilizando las radios MDEK1001.

El resultado obtenido parece un buen punto de partida para llegar al objetivo final el cual es implantar este desarrollo en animales reales, aunque para eso queda bastante recorrido, y el siguiente paso a realizar sería la simulación utilizando robots físicos, el cual parece posible implementar utilizando el desarrollo realizado en este trabajo.

Personalmente este proyecto me ha ayudado bastante desde un punto de vista académico debido a que he podido aprender y trabajar con nuevas tecnologías y sofware como son las máquinas de estados, árboles de comportamiento, Robocomp, etc.

Me ha parecido un trabajo bastante interesante debido a que he podido comprobar

como el uso de la tecnología puede ayudar al sector primario y simular, en este caso, el comportamiento de animales de la forma más realista posible con las herramientas utilizadas y espero que en un futuro pueda ayudar a que este proyecto se acabe desarrollando en su totalidad.

Capítulo 8

Glosario de términos

En este capítulo se explicarán algunos términos utilizados durante la documentación, para así hacer mas sencillo el entendimiento del proyecto.

[1]Evento: aparición de un acontecimineto, ubicado en el tiempo y espacio. Son utilizados en las máquinas de estados para modelar la aparición de un estímulo/señal que haga transicionar al objeto de un estado a otro.

[2]Script: documento de texto plano en el cual se dictan unas instrucciones en un lenguaje de programación las cuales compondrán un algoritmo que compondrá al programa.

[3]Ice: es un middleware orientado a objetos que ofrece llamadas a procedimiento remotos, servicios de publicación y suscripcion y orientación a objetos entre otros, que trabaja bajo la licencia GPL el cual funciona en los principales sistemas operativos.

[4]CDSL: lenguaje de definición de componentes utilizado para crear y definir componentes en Robocomp.

[5]XML: lenguaje de marcado el cual se utiliza en este proyecto para crear los ficheros de configuración para los mapas que se quieren simular con el simulador.

[6]Señal: representa la acción de enviar de manera asíncrona información para que un objeto lo reciba.

[7]Tick: señal que manda la raíz del árbol de comportamiento hacia los diferentes nodos para que se ejecuten.

[8]CPU: unidad central de procesamiento. Es la parte hardware de un componente que se encarga de interpretar las instrucciones de un programa informático

[9]Densidad espectral: función matemática que nos muestra la distribución de la potencia o energía de una señal.

[10]Comunicación Sin-Línea-de-Visión: término utilizado para describir un trayecto en el cual existen obstáculos entre la ubicación del transmisor de la señal y de la ubicación del receptor de esta.

[11]Nodo hoja: es el nodo de nivel más bajo del árbol a pesar de ser el que realiza las funciones.

[12]LearnBlock: es una herramienta de programación educativa creado por Robolab.

Bibliografía

[1] **Robocomp** https://github.com/robocomp/robocomp

[2] **Árboles de comportamiento** https://gamasutra.com/

[3] **LearnBlock** https://github.com/robocomp/LearnBlock

[4] **Librería Árboles de comportamiento** https://github.com/arvidsson/BrainTree

[5] **Radios MDEK1001** https://www.digikey.es/product-detail/es/MDEK1001/

[6] **Gazebo** http://gazebosim.org/

[7] **ROS** https://www.ros.org/

[9] **Getting started with LaTeX** https://www.maths.tcd.ie/

[10] **Repositorio del proyecto** https://github.com/ismaasanchez/robotsOvejas.git

[11] **Overleaf** https://www.overleaf.com/