Tesis de Licenciatura

Nueva interfaz de programación de robots móviles para talleres de Robótica Educativa

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires.

Alumno:

Javier Caccavelli jcaccav@dc.uba.ar

Directores:

Lic. Sol Pedre spedre@dc.uba.ar
Lic. Pablo de Cristóforis pdecris@dc.uba.ar

Índice

Índice de figuras

Índice de cuadros

Agradecimientos

A mis directores y amigos, Sol y Pablo, que me ayudaron mucho y me dieron el impulso para terminar este trabajo.

A Esteban y Sabri, que en el comienzo de la carrera y en este último tiempo me ayudaron mucho con las cursadas, los TPs y los finales. A Seba, que todos los días me recuerda que me reciba.

A mi viejo y mis hermanas, por estar siempre, por su incondicionalidad, sin importar qué. Por bancarme y darme refugio siempre que lo necesité.

A todos los chicos del laboratorio de robótica, a Thomas, Carlos, Facu, Maxi, Kevin y Mati, que metieron mucha mano en la infinidad de códigos, en los robots y en las aplicaciones, en los cursos, talleres y exposiciones. Sin el trabajo infatigable de ellos este trabajo no podría haber llegado hasta aquí.

A todos los estudiantes y docentes que en estos años defendieron la continuidad de la robótica en la facultad y lograron derrotar la entrega del Konabot. Al Taller de Róbotica y todos los chicos que pasaron por él y le dieron un nuevo impulso a la robótica en la facultad. A Andrea, Andrés, Diego, Sol y Pablo por jugárselas también en ese momento difícil para el laboratorio. A Marta, Patricia y Julio por todo lo que hicieron desde ese momento hasta ahora y siguen haciendo. A Andrés, por su compromiso, por ayudarnos siempre, en especial con el ExaBot y el CheBot, por no querer guardarse nada para él, por compartirlo todo.

A los chicos de la Lista Unidad, de la CEPA, por mostrarme que una universidad distinta en un país distinto es posible, por darme las herramientas y por luchar incansablemente por conquistar una universidad verdaderamente democrática, científica y popular, la universidad del pueblo liberado. A todos los estudiantes que en este camino han luchado, y siguen haciéndolo, por la universidad pública, contra la LES y la CONEAU, por el presupuesto, cuestionando el carácter de sus contenidos y confluyendo con los distintos sectores populares.

A todos mis compañeros, en especial aquellos con los que he tenido la suerte de compartir la lucha, de estar hombro a hombro, aquellos que me gustaría tener a mi lado en cualquier batalla. A la Juventud, por su imprescindible y constante aporte en todos los planos, a quienes se entregan abnegadamente a esta empresa cotidiana que es la revolución.

1. Resumen

La Robótica Educativa se propone utilizar la robótica como un recurso didáctico que permita a los alumnos construir su propio conocimiento, de una forma práctica y movilizadora, donde sean creadores e investigadores y no sólo consumidores de conocimientos. Uno de los objetivos de la Robótica Educativa es ayudar a los alumnos en la construcción de sus propias representaciones y conceptos de la ciencia y la tecnología, a través del uso, manejo y control de entornos robotizados. Para lograr este objetivo es fundamental contar con una interfaz adecuada que permita a los alumnos sin experiencia previa interactuar con robots de una manera rápida y sencilla.

En este trabajo presentamos el desarrollo del ERBPI (Easy Robot Behaviour Programming Interface), una nueva interfaz de programación de robots móviles que no requiere del usuario ninguna experiencia previa en programación ni robótica. Para lograr esto, abandonamos el paradigma de programación imperativa y adoptamos un enfoque basado en comportamientos. De esta forma, la nueva aplicación hace uso del paradigma conexionista, donde los comportamientos se definen estableciendo conexiones configurables entre los sensores y actuadores del robot. Por otra parte, los distintos comportamientos se pueden conectar entre sí mediante una arquitectura de subsunción. Esta arquitectura se representa mediante una máquina de estados finitos, donde cada estado representa un comportamiento y cada transición un cambio de comportamiento. Dichas transiciones se activan cuando se cumplen determinadas condiciones asociadas a las acciones y percepciones del robot o a ciertas condiciones de contexto.

El ERBPI está diseñado para trabajar con diferentes tipos de robots y simuladores, abstrayendo los distintos sensores, actuadores, comandos y protocolos de comunicación que posee cada plataforma, haciendo posible añadir fácilmente nuevos robots o simuladores. Las experiencias didácticas realizadas en distintos cursos y talleres con alumnos de la escuela media nos han permitido poner a prueba el ERBPI como una valiosa herramienta didáctica para la Robótica Educativa.

palabras clave: robótica educativa, interfaz de programación de robots, robótica basada en comportamientos.

Abstract

Educational Robotics uses robots as a didactical resource to help students build their own knowledge, in a practical and moving way, being creative and explorers instead of only consumers of knowledge. One of the aims of Educational Robotics is to help students build their own representations and concepts of science and technology by using and controlling robotic environments. To achieve this goal it is vital to have an adequate interface that allows inexperienced students to interact with robots in an quick and easy manner.

In this work we present the development of ERBPI (Easy Robot Behaviour Programming Interface), a new interface that does not require any previous programming or robotic experience to program mobile robots. To accomplish this, we abandon the imperative programming paradigm and take a behaviour-based approach. The new application takes ideas from the connectionist paradigm, defining behaviours by establishing configurable connections between sensors and actuators. Moreover, different behaviours can be connected using a subsumption architecture. This architecture is modeled using a state machine, where each state represents a behaviour and each transition a change of behaviour. Transitions are activated when particular conditions associated to the environment or the robot's actions or perceptions are met.

ERBPI is designed to work with a variety of robots and simulators, abstracting the different sensors, actuators, commands and communication protocols of each robotic platform, making it easy to add new robots or simulators. Didactical experiences in different courses and workshops with middle school students enabled us to test ERBPI as a valuable tool for Educational Robotics.

keywords: educational robotics, robot programming interface, behaviour-based robotics.

2. Introducción

La Robótica Educativa se propone utilizar la robótica como un recurso didáctico que permita a los alumnos construir su propio conocimiento, de una forma práctica y movilizadora, donde sean creadores e investigadores y no sólo consumidores de conocimientos [1] [2]. Uno de los objetivos de la Robótica Educativa es ayudar a los alumnos en la construcción de sus propias representaciones y conceptos de la ciencia y la tecnología, a través del uso, manejo y control de entornos robotizados. En estas experiencias se utiliza el proceso constructivista de diseño, construcción, programación y testeo de los comportamientos del robot, así como la colaboración y trabajo en equipo, como medio eficaz de animar la educación [3].

En los últimos años la Robótica Educativa ha crecido sustancialmente en algunos países. La disponibilidad de acceso a robots y plataformas de programación ha llevado a la utilización de robots en las escuelas de nivel medio [4]. Muchos docentes tienen especial interés en la introducción de robots en sus cursos para la enseñanza de una variedad de temas que no sean específicamente de robótica. Así, la Robótica Educativa se propone el uso de la robótica como un recurso didáctico que permite a los alumnos sin experiencia acercarse a diferentes campos científicos, tales como las matemáticas, ciencias experimentales, tecnología, ciencias de la información y la informática, entre otros [5].

El Laboratorio de Robótica y Sistemas Embebidos (LRSE) del Departamento de Computación, FCEyN-UBA, ha desarrollado diversas actividades, cursos y experiencias en Robótica Educativa en conjunto con la Dirección de Orientación Vocacional de la FCEyN-UBA. Estas actividades estuvieron orientadas a alumnos de los últimos años de la escuela media para despertar el interés por las ciencias y la tecnología y fomentar la promoción de la carrera de Ciencias de la Computación [6] [7] [8].

En la mayoría de estas actividades, se utilizó el modelo teórico de Braitenberg [9] para que los participantes pudieran programar diversos comportamientos en los robots a partir de establecer conexiones entre los sensores y actuadores. También se motivó a los alumnos a vincular los comportamientos alcanzados con características humanas y/o animales y a ejercitar el método científico, es decir, a partir de un problema dado (darle un determinado comportamiento al robot) elaborar hipótesis para resolverlo, programar al robot siguiendo las hipótesis, experimentar y contrastar con la realidad y finalmente sacar conclusiones para mejorar la solución propuesta.

Durante los cursos y talleres de robótica educativa realizados por el LRSE, los participantes pudieron generar, mediante mecanismos muy sencillos, comportamientos complejos aplicables a robots autónomos. En pocos minutos, fue posible desarrollar criaturas capaces de seguir la luz, evitar obstáculos, seguir una línea en el piso o resolver laberintos. De esta forma se validó el modelo teórico de Braitenberg como forma de programar comportamientos para robots móviles sin necesidad de conocimientos previos de programación por parte de los alumnos. Los participantes pudieron utilizar simuladores para generar diferentes tipos de comportamientos y posteriormente implementarlos y ejecutarlos en robots reales. Para dichas experiencias se utilizaron los robots móviles ExaBot

[10] y Khepera [11], y los simuladores ExaSim (simulador del robot ExaBot) [12] y Yaks (simulador del robot Khepera) [13].

La mayor dificultad que ha tenido que afrontar el LRSE a la hora de desarrollar las actividades de Robótica Educativa ha sido la falta de una aplicación de software adecuada. Una interfaz de programación de robots que permitiera establecer una interacción amigable entre los alumnos y los robots, que pudiera ser fácilmente utilizada sin necesidad de conocimientos previos de programación ni de robótica. Al mismo tiempo, esta interfaz de programación no podía ser dependiente de una plataforma, simulador o robot particular, sino que debía permitir trabajar con diversos robots o simuladores, encapsulando sus características específicas.

Así llegamos a abordar uno de los problemas claves a resolver por la Robótica Educativa: contar con una interfaz adecuada entre el público inexperto y los robots, que se conciben en general como máquinas extremadamente complejas [5]. La mayoría de las interfaces existentes para la programación de robots están implementadas como extensiones de los lenguajes de programación más conocidos como *C*, *C++*, *Java* y *Python* [14] [15] [16] [17] [18], por lo que requieren de la experiencia o el interés por parte del usuario en aprender un lenguaje de programación y manejar conceptos de programación imperativa como variables, condiciones de control, ciclos o iteraciones en un programa. Además, en su mayoría, estas interfaces basadas en lenguajes de programación son específicas para una determinada plataforma, en particular para los robots del kit Lego Mindstorms. También existen varios entornos gráficos de simulación de robots destinados a la escuela media [19] [20] [21], que si bien son interfaces gráficas, de muy fácil manejo para alumnos sin experiencia, sólo permiten desarrollar experiencias en simulación, perdiendo de vista la interacción con robots reales. Además, y pese a ser más intuitivas, estas interfaces mantienen la influencia de la programación imperativa.

En este trabajo presentamos el desarrollo del ERBPI (Easy Robot Behaviour Programming Interface), una nueva interfaz gráfica de programación de robots, fácil de aprender y de usar, y que puede ser utilizada por personas sin conocimientos previos de programación ni de robótica. La interfaz fue desarrollada para adaptarse a los cursos y talleres de Robótica Educativa del LRSE ya existentes y al mismo tiempo permitir que nuevas actividades pudieran llevarse a cabo con mayor facilidad y flexibilidad. Además, está diseñada para facilitar la capacitación de docentes y divulgadores sin conocimientos de robótica, para que puedan llevar adelante los cursos y talleres, multiplicando el alcance de las experiencias didácticas.

Entre los requerimientos que guiaron el desarrollo de la aplicación podemos citar:

- *Fácil de usar:* El usuario no debe requerir ningún conocimiento previo de programación ni robótica. La interfaz debe ser intuitiva y fácil de aprender y usar, proporcionando todas las herramientas para definir los comportamientos del robot de forma gráfica.
- *Independencia de la Plataforma:* La aplicación debe utilizarse con una gran variedad de robots y simuladores y ser fácilmente extensible para

controlar nuevos robots y simuladores. Por otra parte, los usuarios deben ser capaces de probar y ejecutar los mismos comportamientos en múltiples robots de manera sencilla.

- *Portabilidad:* La aplicación debe ser capaz de utilizarse en diferentes sistemas operativos y plataformas para adaptarse a los diversos recursos de hardware y software disponibles en las instituciones educativas.
- *Flexibilidad:* La interfaz debe poder ser utilizada eficientemente por una amplia gama de alumnos y docentes de distintos niveles educativos, necesidades curriculares, materias y contextos.

Para lograr estos objetivos, abandonamos el paradigma de programación imperativo y encapsulamos la programación de bajo nivel de control de los robots. La interfaz propuesta está enfocada en la programación de comportamientos (que llevará adelante el robot) y no de un algoritmo que se ejecute paso a paso.

Siguiendo este enfoque, la nueva interfaz se basa en el paradigma conexionista, donde los comportamientos se definen estableciendo conexiones configurables entre los sensores y actuadores del robot, siguiendo el modelo propuesto por Braitenberg. Al mismo tiempo, distintos comportamientos definidos de esta forma pueden conectarse entre sí mediante una arquitectura de subsunción [22], alcanzando comportamientos más complejos. Esta arquitectura puede representarse mediante una máquina de estados finitos. Así, se obtiene un modelo donde cada estado representa un comportamiento (definido a partir de las conexiones entre los sensores y actuadores del robot) y cada transición entre estados, un cambio de comportamiento, que se activa cuando se cumplen determinadas reglas asociadas a las acciones y percepciones del robot o a ciertas condiciones de contexto.

Como la aplicación se desarrolló dentro de la universidad pública, está íntegramente implementada siguiendo los lineamientos de Software Libre. Esto significa que todos los usuarios tienen libertad para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Como se pretende la mayor portabilidad para poder llevar la aplicación a distintos ámbitos como talleres en escuelas, charlas y demostraciones en diferentes instituciones, se utilizaron lenguajes de programación estándar como *C*, *C++* y *Java*, que pueden ser compilados y ejecutados en los sistemas operativos más comunes como Windows y Linux.

El trabajo se llevó a cabo en el LRSE, que cuenta con diversos robots y simuladores que sirvieron como plataforma para el desarrollo y testeo de la aplicación. Además, diferentes versiones de la aplicación fueron utilizadas en los Talleres de Robótica: "Talleres de Robótica Educativa para alumnos de escuela media" en el marco de los proyectos de extensión Exactas con la Sociedad de la FCEyN-UBA, durante las Semanas de la Computación organizadas por el Departamento de Computación de la FCEyN-UBA y en otras actividades de divulgación y extensión como los "Talleres de Ciencias" organizados por la Dirección de Orientación Vocacional de la FCEyN-UBA. Estas experiencias sirvieron para probar la eficacia de la interfaz, y también para recibir ideas y críticas de los alumnos y mejorarla. Es nuestra intención continuar utilizando y mejorando la aplicación en futuros talleres y actividades de Robótica Educativa.

El resto del informe se organiza de la siguiente forma: en el siguiente capítulo introducimos los conceptos preliminares, en el capítulo 4 describimos algunos trabajos relacionados, en el capítulo 5 detallamos el diseño e implementación de la nueva interfaz de programación de robots propuesta, en el capítulo 6 presentamos una formalización de la aplicación, que permite definir los alcances y límites desde el punto de vista del poder de expresividad del lenguaje, en el capítulo 7 presentamos los resultados del trabajo y finalmente concluimos en el capítulo 8 delineando algunas propuestas a futuro.

3. Preliminares

3.1. Robótica Educativa

Como fue dicho anteriormente, la Robótica Educativa se propone utilizar la robótica como un recurso didáctico. Como disciplina pedagógica, tiene un modelo aún en desarrollo. De hecho su nomenclatura tiene aún diversas variantes como *Robótica Pedagógica*, *Robótica y Educación* y *Robótica Aplicada a la Educación*. Sin embargo, el método didáctico más utilizado por la Robótica Educativa es el constructivista, basado en el diseño, construcción, programación y testeo de los comportamientos del robot, así como la colaboración y trabajo en equipo, como medio eficaz de animar el proceso educativo [23].

A modo de síntesis, se puede decir que la Robótica Educativa es una disciplina con los siguientes lineamientos:

- No es un intento de enseñanza a los alumnos en el área de la robótica, sino aprovechar la interdisciplinariedad que la compone, para activar procesos cognitivos y sociales que propicien un aprendizaje significativo, permitiendo de esta manera el desarrollo del pensamiento y un acercamiento provechoso al mundo de la ciencia y la tecnología [2].
- No es un estudio teórico-práctico sobre robots. Tampoco es una actividad de juego con robots. Consiste, más bien, en la generación de entornos tecnológicos que permitan a los alumnos la integración de distintas áreas del conocimiento para la adquisición de habilidades generales y de nociones científicas, involucrándose en un proceso de resolución de problemas con el fin de desarrollar en ellos un pensamiento sistemático, estructurado, lógico y formal [1] [2].
- Los entornos de aprendizaje generados por la Robótica Educativa deben estar basados fundamentalmente para la acción de los alumnos. Son los alumnos quienes deberán concebir, construir, programar y poner en funcionamiento los robots que les permitan resolver los problemas propuestos. En este proceso desarrollan los diversos conocimientos científicos y tecnológicos, favoreciendo el desarrollo de la iniciativa, la creatividad, el trabajo en equipo, y el interés por la investigación [2] [3] [23].
- Posibilitar el desarrollo de la noción *causa-efecto* al ofrecer un espacio para la observación, exploración y reproducción de fenómenos reales precisos [3] [23].
- Otorgar a los alumnos la libertad de cometer errores. El error es una fuente de aprendizaje que puede usarse para mejorar la comprensión de los problemas. Darle la mayor libertad en la toma de decisiones a cada alumno, donde el docente es un facilitador que le muestra alternativas en función de las soluciones propuestas por ellos, supone aprender a partir de los errores que ellos mismos tienen que corregir, elaborando y validando los experimentos mediante el método científico [2] [23].
- Fomentar el aprendizaje del proceso científico, de representación y modelado matemático, lógico y formal, así como también el desarrollo de

habilidades en los procesos de análisis y síntesis mediante la formulación de hipótesis, los resultados obtenidos y la elaboración de conclusiones [2].

De esta forma, la Robótica Educativa resulta de una utilidad didáctica muy amplia, admitiendo diversas formas de utilización según los objetivos y la currícula, ya que permite a los docentes y alumnos modificar su contenido y adaptarlo a sus necesidades concretas.

La Robótica Educativa ha crecido sustancialmente en los últimos años. La disponibilidad de acceso a robots y plataformas de programación ha llevado en algunos países a la utilización de robots en las escuelas medias. Muchos docentes tienen especial interés en la introducción de robots en sus cursos para la enseñanza de una variedad de temas que no son específicamente de robótica [3] [5] [24]. Existen también diversos congresos, conferencias y revistas que abordan esta temática.

Sin embargo, el reto actual de la Robótica Educativa sigue siendo pasar de ser una actividad extracurricular a integrarse como recurso didáctico dentro de la currícula escolar de forma permanente, no sólo en las asignaturas tecnológicas sino también en aquellas donde pueda servir como apoyo para mejorar los procesos de enseñanza y aprendizaje en otras disciplinas [2]. En la actualidad, una de las limitaciones más importantes con la que se enfrenta la Robótica Educativa para cumplir con este objetivo, y que condiciona su desarrollo, es la disponibilidad de herramientas para la programación fácil de los robots, documentación de apoyo y disponibilidad económica de acceso a estas herramientas. Por eso, el grado de desarrollo que la Robótica Educativa podrá adquirir dependerá en gran medida del acceso y capacitación que, tanto los docentes como alumnos, puedan obtener a herramientas, interfaces de programación y plataformas robóticas para poder realizar las experiencias [2] [25].

3.2. Vehículos de Braitenberg

Un vehículo de Braitenberg es un modelo conceptual de un robot móvil. Estos vehículos permiten desarrollar experimentos para ilustrar evolutivamente las capacidades de distintos agentes simples o criaturas [9] y analizar los comportamientos emergentes. Se componen de tres elementos:

- **Ruedas:** generalmente dos, una a la izquierda y otra a la derecha.
- **Sensores:** de diferentes tipos que permiten percibir distintas características del ambiente como objetos, luz, calor, etc.
- **Conexiones:** entre los sensores y las ruedas.

En la Figura 1 puede verse una ilustración de dos ejemplos de estos vehículos.

En una primera instancia, las conexiones entre los sensores y ruedas pueden ser excitatorias, representadas con el símbolo "+" (cuanto más percibe el sensor más energía transfiere a la rueda) e inhibitorias representadas con el símbolo "−" (cuanto más percibe el sensor menos energía transfiere a la rueda). Por lo tanto, los vehículos de la Figura 1 formados por conexiones excitatorias e

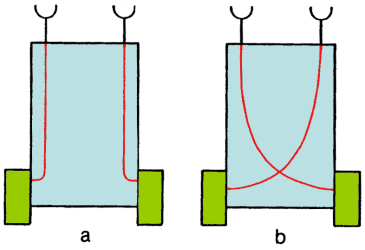

Figura 1: Ilustración del *Vehículo 2* de Braitenberg. En el *Vehículo 2a* las conexiones de los sensores son con las ruedas del mismo lado. En el *Vehículo 2b* las conexiones son cruzadas.

inhibitorias, resultan en agentes reactivos bastante simples. Sin embargo, cuando estos vehículos son colocados en un entorno particular, pueden exhibir un comportamiento más interesante.

En la Figura 2 podemos ver dos ejemplos de vehículos con conexiones excitatorias. Además tomaremos para el ejemplo que poseen sensores de luz y el objeto en el centro es una fuente lumínica. En el primer caso, *Vehículo 2a*, puede apreciarse que su comportamiento será el de avanzar esquivando la fuente de luz. Mientras que en el otro caso, *Vehículo 2b*, su comportamiento será el de avanzar girando hacia la misma. Sin embargo, si el vehículo se encuentra orientado hacia la luz, de modo que ambos sensores transfieran la misma energía a cada rueda, ambos vehículos se estrellarán contra la fuente de luz. A modo de ejemplo, Braitenberg realiza interpretaciones de estos comportamientos. A ambos vehículos les molesta la fuente de luz, pero ambos reaccionan diferente ante la misma. El *Vehículo 2a* intenta escapar cobardemente, mientas que el *Vehículo 2b* responde agresivamente yendo directo a ella.

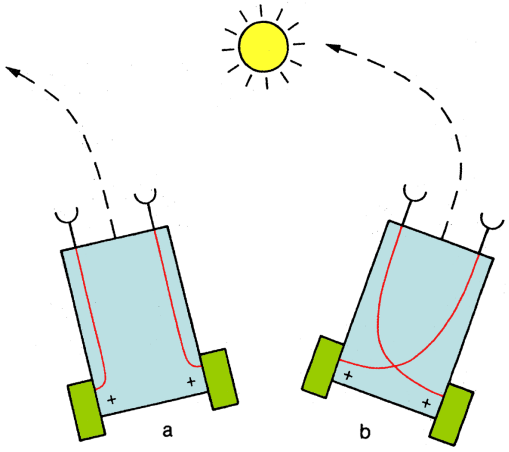

Figura 2: Ilustración del comportamiento del *Vehículo 2* de Braitenberg con conexiones excitatorias.

En la Figura 3 podemos ver el mismo ejemplo, pero esta vez con conexiones inhibitorias. En el primer caso, *Vehículo 3a*, puede apreciarse que su comportamiento será el de avanzar girando hacia la fuente de luz y, al acercarse, detenerse cuando se encuentre apuntando directo a ella. Mientras que en el otro caso, *Vehículo 3b*, su comportamiento será el de avanzar hacia la fuente de luz y, al acercarse, detenerse intentado esquivarla. Una interpretación a modo de ejemplo de estos comportamientos podría ser que a ambos vehículos les agrada la fuente de luz. Sin embargo, el *Vehículo 3a* se queda concentrado enfocándola, mientas que el *Vehículo 3b* le agrada estar cerca de la fuente de luz pero se mantiene enfocado hacia otro lado a la espera de una fuente mejor.

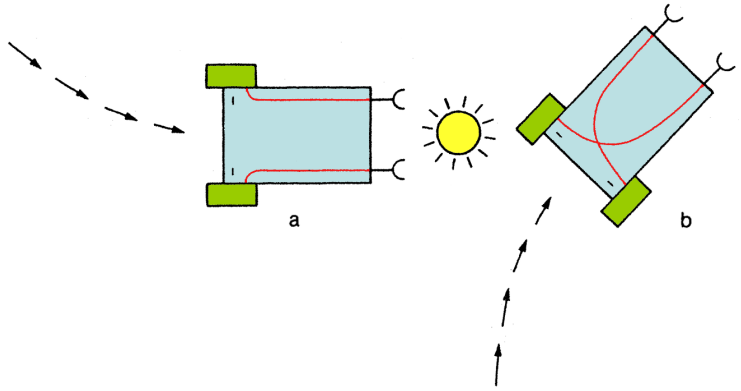

Figura 3: Ilustración del comportamiento del *Vehículo 3* de Braitenberg con conexiones inhibitorias.

Un vehículo de Braitenberg puede describirse como un agente reactivo que puede moverse de forma autónoma en un entorno dado. Dependiendo del entorno y de cómo se establecen las conexiones entre los sensores y las ruedas, el vehículo puede exhibir comportamientos diferentes e incluso adaptativos, es decir, comportamientos que intentan lograr ciertas situaciones y evitar otras, ajustándose a los cambios en el ambiente. Otro aspecto importante del modelo de Braitenberg es que el funcionamiento del vehículo es puramente reactivo, esto quiere decir que el comportamiento emerge exclusivamente en función de los valores actuales de los sensores, sin ningún tipo de procesamiento de la información sensada, ni representación del modelo del mundo, o conocimiento de estados anteriores o futuros.

En una segunda instancia, los vehículos pueden ser más complejos, conteniendo un número mayor de sensores de distintos tipos y la posibilidad de utilizar conexiones excitatorias e inhibitorias simultáneamente, dando lugar a los *vehículos multisensoriales*. En la Figura 4 puede verse un ejemplo de este vehículo. En este ejemplo, Braitenberg interpreta el comportamiento de este vehículo como un agente al que no le gusta el calor, huye de los lugares calientes. Se comporta de forma agresiva con las fuentes de luz, lo que tiene sentido ya que son generadores de calor. Le agrada el oxígeno y la materia orgánica, si escasea el oxígeno se mueve hacia un lugar con más oxígeno.

Por último, Braitenberg introduce la posibilidad de conexiones más complejas entre sensores y ruedas. En particular, se introducen funciones matemáticas

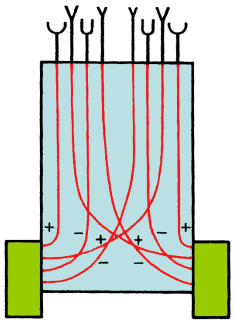

Figura 4: Ilustración del *Vehículo 3c* multisensorial de Braitenberg. Posee cuatro pares de sensores: sensores de luz, de temperatura, oxígeno y materia orgánica. Las conexiones son de distintos tipos: excitatorias sin cruzar, excitatorias cruzadas, inhibitorias sin cruzar e inhibitorias cruzadas, respectivamente.

como un nuevo tipo de conexión más general. Un ejemplo de función a utilizar es la función partida. De esta forma, se pueden describir comportamientos aún más complejos. En la Figura 5 puede verse un ejemplo de esta función. Así un sensor puede transferir energía a una rueda dependiendo del escalón de la función. Por ejemplo, podría casi no transferir energía dentro de un rango, comportarse proporcionalmente dentro de otro y finalmente transferir mucha energía dentro del último.

Este nuevo tipo funciones matemáticas parametrizables en las conexiones, sumado a los vehículos multisensoriales y a la posibilidad de establecer múltiples conexiones entre sensores y ruedas, expande ampliamente la posibilidad de definir comportamientos cada vez más complejos.

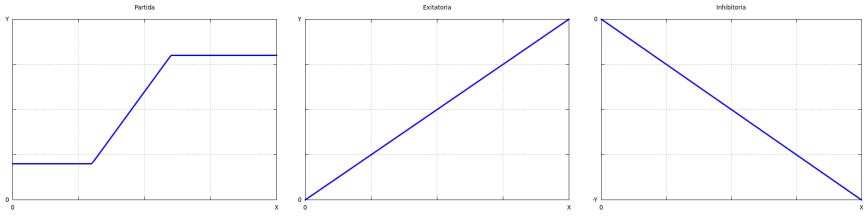

Figura 5: Esquema de la función partida. Las funciones excitatorias e inhibitorias pueden verse como un caso particular de esta función.

En adelante, nos referiremos al *modelo de los Vehículos de Braitenberg* como el esquema que representa el comportamiento de un robot en función de su conjunto de sensores, ruedas y las funciones matemáticas que los conectan entre sí. También nos referiremos a las ruedas como *actuadores* como concepto más general. En la Figura 6 puede verse un ejemplo de este esquema.

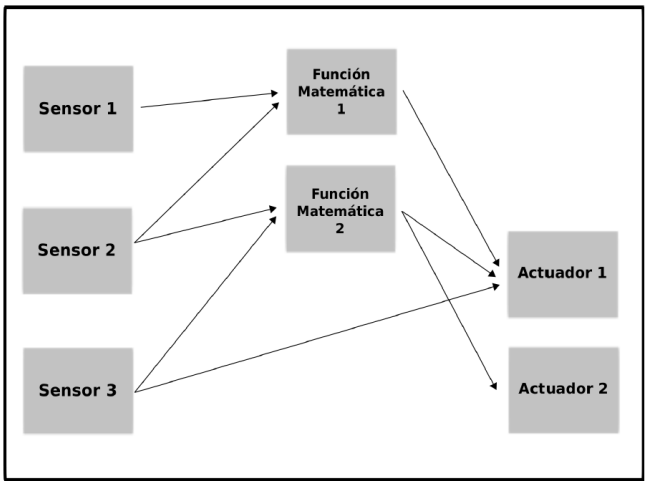

Figura 6: Esquema de conexiones entre sensores, funciones y actuadores siguiendo el modelo de Vehículos de Braitenberg.

3.3. Arquitectura de Subsunción Basada en Comportamientos

La arquitectura de subsunción (*subsumption architecture*) [26] [27] se desarrolla como una técnica para facilitar el diseño de algoritmos en robots que tuvieran múltiples objetivos respondiendo a múltiples sensores de una forma reactiva. Se buscaban arquitecturas robustas frente a fallos, en las que los robots tuvieran como prioridad su supervivencia evitando situaciones de desperfectos, contemplada en las capas de bajo nivel en la arquitectura. Pero, que a su vez, fuese una arquitectura modular y extensible, para que los robots fueran capaces de realizar cada vez tareas más complejas. Esta última característica es la que convirtió a la arquitectura de subsunción en una de las técnicas más utilizadas para la programación de robots reactivos o autónomos basados en comportamientos, ya que se comienza construyendo robots con comportamientos simples y luego se construyen comportamientos más complejos sobre la base de la combinación e inclusión de ellos sin la necesidad de realizarles modificaciones [28].

La arquitectura de subsunción es una forma de descomponer el comportamiento complejo de estos robots en varios módulos o comportamientos simples. Cada módulo implementa un objetivo particular del robot. El concepto de subsunción surge del proceso de coordinación entre los módulos. Los módulos más complejos subsumen o incluyen a los más simples. Existe una jerarquía de prioridades entre los distintos módulos. Los módulos de alto nivel subsumen a los módulos de bajo nivel. Así, se proveen las bases para un diseño incremental de los comportamientos.

La arquitectura de subsunción es una de las precursoras de lo que actualmente se conoce como arquitecturas basadas en comportamientos (*behaviour-based architecture*) [22]. Este enfoque surge como contrapuesto al que se utiliza en las arquitecturas clásicas basadas en el conocimiento. Estas últimas se organizan de forma secuencial. Existe una primera fase de sensado, donde la

información del entorno es integrada progresivamente dentro de un modelo o representación interna del mundo. Luego son evaluadas las posibles siguientes acciones y se planifica cuáles llevar a cabo. Finalmente se ejecutan las acciones seleccionadas en base a la planificación. La principal crítica que se hace a este enfoque clásico utilizado por las arquitecturas basadas en el conocimiento, es la necesidad de contar con un modelo del mundo, que siempre será impreciso e incompleto. Además, por la necesidad de integrar y procesar la información almacenada antes de tomar una decisión, este tipo de arquitectura presenta una baja velocidad de respuesta frente a cambios bruscos en el ambiente.

En contraste, en las arquitecturas basadas en comportamientos se intenta evitar el modelo del mundo, reduciendo o eliminando completamente la fase de planificación para la toma de decisiones. Por lo tanto, el comportamiento del robot es más rápido frente a los estímulos y modificaciones del entorno, obteniendo en consecuencia comportamientos reactivos. Las técnicas de programación basadas en comportamientos son técnicas para la toma de decisiones de acciones en robots autónomos, es decir, robots que pueden realizar una determinada tarea en entornos desconocidos sin la intervención de un humano. Sin embargo, un problema que presentan las arquitecturas basadas en comportamientos es la dificultad para desarrollar tareas que intrínsecamente requieran de cierta planificación o almacenar la historia de los estados del sistema, por lo que la tendencia en la actualidad es realizar enfoques mixtos entre arquitecturas basadas en comportamientos y en el conocimiento [29].

Existen varias formas de representar la toma de decisiones en una arquitectura de subsunción. La más utilizada es mediante la definición de reglas asociadas a una acción y una condición. Entonces, se puede expresar esta decisión de la forma: *si se cumple la condición C, entonces realizar la acción A*, o también como: *si la condición C se sigue cumpliendo, seguir realizando la acción A*.

Una forma de representar la arquitectura de subsunción es mediante las máquinas de estados finitos (FSM), donde cada estado representa un módulo o comportamiento simple y las transiciones entre estados las reglas asociadas a las acciones y condiciones [22]. En la Figura 7 puede verse un ejemplo donde se describe el comportamiento de un robot explorador. Los robots exploradores son robots autónomos móviles capaces de explorar un entorno desconocido evitando los obstáculos que pudieran interponerse en su trayectoria.

Un ejemplo para este comportamiento descripto en la Figura 7 es interpretar que el robot recarga sus baterías con energía solar y, por lo tanto, debe moverse en el entorno siempre y cuando tenga batería suficiente para hacerlo. De lo contrario, debe buscar una fuente de luz para recargar su batería evitando los obstáculos que pudieran presentarse en su camino. Una vez recargada, puede seguir moviéndose para explorar. En ambos casos, los comportamientos simples para *moverse* o *recargar batería*, deben *evitar obstáculos* en el caso de necesitarlo. De esta forma, queda determinada la arquitectura de subsunción entre los distintos módulos o comportamientos simples en el ejemplo. El módulo *moverse* subsume al módulo *recargar batería* y el módulo *recargar batería* subsume a *evitar obstáculo*, determinando el comportamiento complejo *explorar*.

También suelen utilizarse para la representación máquinas de estados finitos ampliadas (AFSM) con la posibilidad de incluir temporizadores (*timers*) para

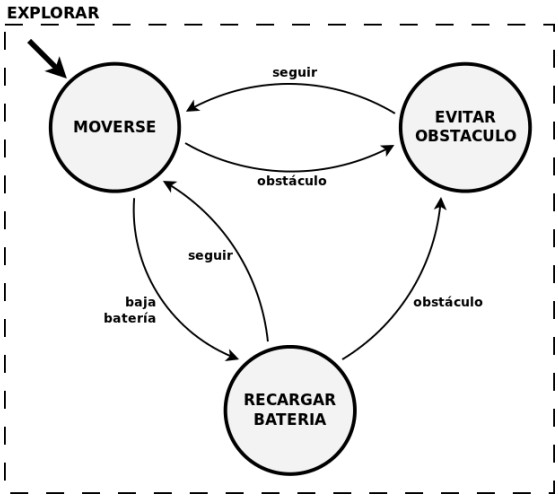

Figura 7: Ilustración de máquina de estados finitos de un robot explorador. De la arquitectura se desprende el proceso de coordinación o subsunción entre los distintos módulos, donde el de más alto nivel *moverse* subsume a *recargar batería*, y el módulo *recargar batería* subsume al de más bajo nivel *evitar obstáculo*, determinando el comportamiento complejo *explorar*.

condicionar el cambio entre estados [22].

3.4. Robots y Simuladores

Existen diversas definiciones de robot, pero para los fines de la Robótica Educativa resulta más atractivo el concepto de robot autónomo. Actualmente existe un consenso en entender por robot autónomo o "inteligente" a una máquina capaz de extraer información de un entorno mediante sensores y utilizar esa información para actuar en consecuencia, sin la intervención de un humano, de forma que obedezca a sus propósitos y objetivos. [22].

El LRSE cuenta con diversos robots y simuladores que sirven como plataformas robóticas para el desarrollo de investigación, docencia y extensión.

Entre los más utilizados para el desarrollo de actividades de extensión y docencia, podemos citar al robot ExaBot, el simulador ExaSim (simulador del robot ExaBot), el robot Khepera y el simulador Yaks (simulador del robot Khepera).

3.4.1. ExaBot

El robot ExaBot fue diseñado en el LRSE para ser utilizado en tareas de investigación, educación y popularización de la ciencia [10] [30]. Como el robot fue pensado para esta variedad de aplicaciones, fue diseñado para poder ser reconfigurado en su capacidad de sensado y procesamiento.

El ExaBot incluye una amplia variedad de sensores, como un anillo de telémetros infrarrojos y un sonar para medir distancias, sensores de detección de línea, sensores de contacto, sensores de movimiento en sus orugas (*encoders*), la posibilidad de agregar una cámara de video y también otros sensores.

Además, posee la capacidad de reconfigurar la ubicación de sus sensores para adaptarse a las tareas específicas que se requieran. La variedad y reconfigurabilidad de sus sensores hace al ExaBot un robot muy versátil para utilizarse en varias aplicaciones como los vehículos de Braitenberg, *line tracing*, evitamiento de obstáculos, *foraging*, aprendizaje en robots, sistemas multi-robots y comportamiento colectivo. Esto lo transforma en una excelente plataforma robótica para utilizar en diversos cursos y talleres de Robótica Educativa.

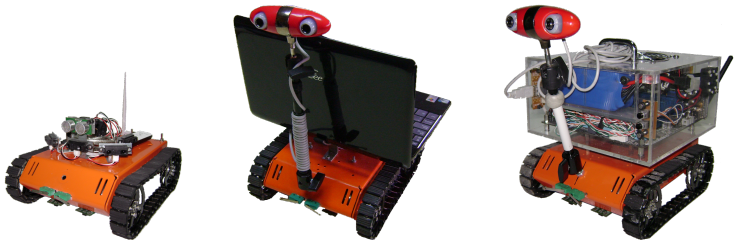

Figura 8: Robot ExaBot, tres de sus configuraciones más utilizadas. De izquierda a derecha, configuración estándar utilizada para el ERBPI, configuración con netbook para visión estéreo con algoritmos de visión, y configuración con módulo de GPU para procesamiento de alto rendimiento de video e imágenes.

La capacidad de procesamiento del ExaBot también es reconfigurable. Entre las configuraciones más utilizadas se encuentran, la configuración estándar con su procesador embebido, la configuración con netbook para visión estéreo con algoritmos de visión, y la configuración con módulo de GPU (Graphics Processing Unit) para procesamiento de alto rendimiento de video e imágenes. Además, su procesador embebido con un sistema operativo Linux de última generación, le proporciona al ExaBot la capacidad de procesamiento autónomo y ejecutar diversas aplicaciones localmente en el robot. En la Figura 8 pueden verse distintas configuraciones posibles del ExaBot. En la actualidad, para realizar las experiencias de Robótica Educativa, la configuración más utilizada es la primera, con el anillo de telémetros, el sonar, los sensores de detección de linea y de contacto, para brindar mayor variedad de sensores para resolver los distintos problemas.

3.4.2. ExaSim

Llamamos ExaSim a una configuración particular del simulador Player/Stage [12] [31] hecha por el LRSE para simular el robot ExaBot. Como en el robot, el ExaSim simula el anillo de seis telémetros infrarrojos para medir distancias. Además, este simulador permite definir los distintos entornos y escenarios de forma rápida y sencilla mediante la utilización de imágenes, pudiendo también simular los mismos en 3D.

El simulador Player/Stage es un proyecto de software libre para generar

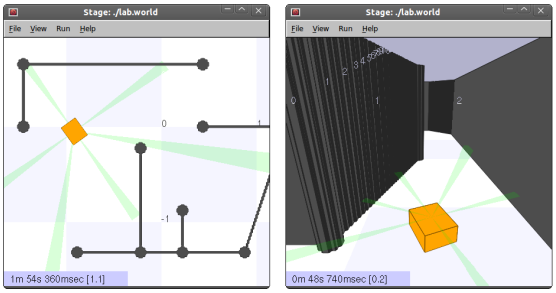

Figura 9: Simulador ExaSim

herramientas para la investigación y desarrollo, y es actualmente uno de los más utilizados en el campo de la robótica con estos fines.

En la Figura 9 pueden verse distintas posibilidades de configuraciones y escenarios para el simulador ExaSim.

3.4.3. Khepera

El robot Khepera es un robot móvil muy pequeño desarrollado a mediados de 1990 [11] [32]. Es uno de los primeros robots móviles con dos ruedas con sensores de movimiento (*encoders*) de su generación. Posee un cuerpo circular de apenas 55mm de diámetro y 30mm de alto con un anillo de ocho pares de sensores infrarrojos de proximidad y de luz ambiente.

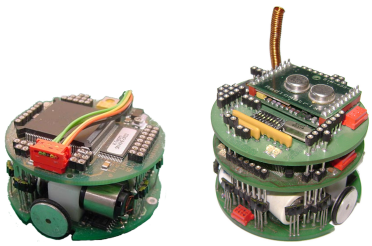

Figura 10: Robot Khepera

En la Figura 10 pueden verse distintas posibilidades de configuraciones para el robot Khepera.

Durante muchos años fue unos de los robots más pequeños, rápidos y económicos, utilizado ampliamente para investigación y desarrollo. Sus cualidades también lo llevaron ser utilizado en diversas actividades no necesariamente en el campo de investigación en robótica, como el desarrollo de actividades de extensión y docencia.

3.4.4. Yaks

En la actualidad existe una gran variedad de simuladores del robot Khepera para distintos lenguajes de programación. Entre otros, podemos citar Khli (Lisp), Webots (Java), Kiks (Matlab) y Yaks (C++).

El simulador Yaks (Yet Another Khepera Simulator) es de código abierto [13], con lo que resulta sencillo poder adaptarlo a los requerimientos específicos. Además, permite separar el programa de control del robot del simulador en sí, lo que permite controlar las simulaciones del robot desde otra aplicación. Posee soporte para el manejo de los distintos elementos del robot de la misma forma que el robot Khepera, de modo que es posible programar una aplicación indistintamente para el robot como para el simulador, intercambiándolos en cualquier momento.

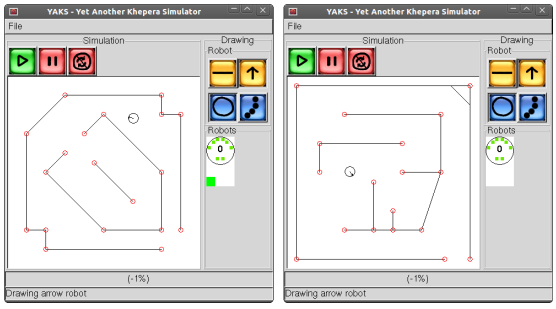

Figura 11: Simulador Yaks

En la Figura 11 pueden verse distintas posibilidades de configuraciones y escenarios para el simulador Yaks.

4. Trabajos Relacionados

Desde la creación del lenguaje de programación Logo con fines didácticos [33] a la actualidad, se han desarrollado una gran diversidad de trabajos en el área de la investigación de la enseñanza en las tecnologías informáticas.

Como un área particular de estas investigaciones, emerge en los últimos años la investigación en el campo de la Robótica Educativa, haciendo hincapié en la interacción entre estas tecnologías y el desarrollo de formas innovadoras de aprendizaje [34] [35].

De esta forma, la Robótica Educativa se ha introducido como una herramienta en la enseñanza y el aprendizaje de las ciencias de gran alcance y potencialidad, una herramienta flexible que permite estimular a los alumnos al controlar el comportamiento de modelos tangibles y hacerlos participar activamente en las actividades de resolución de problemas [36].

El desarrollo de las nuevas tecnologías y la disponibilidad creciente de acceso a ellas, incluyendo plataformas robóticas y de programación, han ayudado al desarrollo de la Robótica Educativa como disciplina y a la utilización de robots en la escuela media [4].

Los robots son máquinas cada vez más complejas que hacen necesario un conocimiento técnico para poder programarlos y controlarlos. En la mayoría de los casos, cuantas mayores capacidades posee un robot, mayor es la complejidad de su diseño y, por lo tanto, la de su control. Además, la mayoría de los robots no poseen interfaces simples para poder utilizarlos. Sus interfaces suelen ser complejas e involucran lenguajes de programación para explotar su funcionalidad. Esto plantea un problema para el uso de los robots por parte del público inexperto.

En los últimos años, el desarrollo de entornos y plataformas de programación de robots desempeña un papel cada vez más importante en la investigación en robótica en general y, en particular, en actividades vinculadas a la Robótica Educativa [37] [38] [39].

Así, han surgido una variedad de interfaces de programación utilizadas en actividades vinculadas a la Robótica Educativa con la idea de facilitar el desarrollo de las aplicaciones y la programación de los robots. Estas interfaces ponen el esfuerzo en proporcionar un modelo para la organización del código y librerías con funcionalidades comunes para facilitar la programación.

Por un lado, existen las interfaces diseñadas para nivel universitario o medio avanzado que están implementadas como extensiones de lenguajes de programación convencionales. Tal es el caso de *Pyro* [14] basada en *Python*, *NQC* [15] basada en *C*, *brickOS* [16] basada en *C++* y *LeJOS* [17] basada en *Java*.

Todas estas interfaces de programación requieren experiencia o interés en aprender un lenguaje de programación en particular. Esto las vuelve inadecuadas para alumnos de la escuela media que no manejan conceptos de programación imperativa o de procedimiento, tales como los conceptos de ciclos, condiciones o variables en un programa. Además, en su gran mayoría, son interfaces de programación para robots específicos, como el kit de Lego Mindstorm.

En el mismo sentido, se encuentra la herramienta *Microsoft Robotics Developer Studio* [40], que incluye una interfaz de programación visual basada en el enfoque de flujo de datos, y nuevamente requiere del conocimiento avanzado de conceptos de programación, por lo que resulta en una herramienta bastante compleja para usuarios inexpertos.

A diferencia de estas interfaces, el objetivo del ERBPI es ser una interfaz gráfica, fácil de aprender y usar, y que no requiera ningún conocimiento previo de programación ni robótica. Para esto, abandonamos el paradigma de programación imperativo y los lenguajes de programación convencionales para adoptar un enfoque basado en comportamientos.

Existen también varias interfaces o entornos gráficos de simulación destinados a las escuelas medias. Este es el caso de *StartLogo* [19], *Squeak Etoys* [20] y *Scratch* [21].

Estas herramientas son interfaces de programación fáciles de usar, que permiten a los alumnos sin experiencia realizar rápidamente un programa. Sin embargo, mantienen la influencia de la programación imperativa por lo que nuevamente son necesarios estos conceptos. Además, sólo están diseñadas para realizar simulaciones por software, con lo que se pierde de vista la interacción de los alumnos con robots en entornos reales.

Otra interfaz de programación ampliamente utilizada es *RoboLab* [18]. En este caso, se trata de un entorno gráfico en el que los alumnos pueden arrastrar y colocar (*drag-and-drop*) sobre un escritorio de trabajo los distintos elementos contenidos en una barra de herramientas. Los iconos representan los componentes del robot, como actuadores y sensores, así como estructuras abstractas de programación como los ciclos y las variables. Si bien esta herramienta aventaja claramente al resto por su facilidad de uso casi completamente en forma gráfica, una vez más, es particular para el kit de Lego Mindstorm y utiliza las estructuras de programación clásicas del paradigma imperativo añadiendo complejidad al entorno del robot.

A diferencia de estas interfaces, uno de los objetivos del ERBPI es estar preparado para ser utilizado con una gran variedad de robots. Al mismo tiempo, que sea fácilmente extensible para incorporar nuevos modelos de robots y simuladores.

Un detalle importante de muchas de las herramientas mencionadas al momento, es que suelen no ser de código abierto o, si lo son, están basadas en utilizar el sistema operativo estándar del kit Lego Mindstorm, que no lo es. Muchas veces, resulta muy difícil adaptar estas herramientas a las necesidades específicas de las experiencias de Robótica Educativa diagramadas, ante la imposibilidad de poder realizarles modificaciones.

El enfoque del ERBPI es ser completamente de código abierto (*open source*) siguiendo los lineamientos de Software Libre, otorgando a los usuarios la libertad para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar la aplicación.

En la gran mayoría de los casos mencionados anteriormente, resulta en una tarea nada sencilla modelar robots o agentes autónomos en entornos particulares. Esto se debe principalmente a que los lenguajes de programación tradicionales no han sido concebidos con esta idea. Por eso, siguiendo los lineamientos

de la arquitectura de subsunción basada en comportamientos, existen trabajos que centran su esfuerzo en proporcionar herramientas de programación para robots que no se basen en el paradigma imperativo de programación.

En este sentido, existen trabajos que desarrollan interfaces de programación de comportamientos de robots utilizando XML (Extensible Markup Language) [41] y centrados en el desarrollo de herramientas que permitan, a usuarios sin conocimientos de programación, la creación fácil y rápida de agentes autónomos [42] [43]. También, trabajos recientes que centran su esfuerzo en generar herramientas independientes de las plataformas robóticas [44] [45]. Sin embargo, como en ocasiones anteriores, estos entornos se basan en el desarrollo de lenguajes de programación específicos DSL (Domain-Specific Language) con lo que se necesita aprender un lenguaje determinado, en algunos casos, o sólo permiten la simulación de los comportamientos definidos, en otros.

En resumen, la nueva interfaz de programación propuesta combina y extiende características encontradas en los trabajos relacionados antes mencionados y agrega características novedosas. El ERBPI está concebido para programar robots autónomos siguiendo el modelo basado en comportamientos. No requiere ningún conocimiento previo de programación ni robótica. Posee una interfaz completamente gráfica, fácil de aprender y usar. Está preparado para ser utilizado con una gran variedad de robots y simuladores y agregar nuevos fácilmente. Su código es abierto (*open source*), siguiendo los lineamientos de Software Libre.

5. Diseño e implementación del ERBPI

5.1. De los requerimientos al diseño

En esta sección detallamos las decisiones de diseño surgidas a partir del análisis de requerimientos, detallados en el Punto 2.

Uno de los requerimientos del ERBPI es que sea *fácil de aprender y usar*, para lo cual *el usuario no debe requerir ningún conocimiento previo de programación ni robótica.*

Para satisfacer este requerimiento, decidimos contar con un módulo específico de la aplicación que sea una interfaz gráfica con el usuario (GUI - Graphical User Interface). Esta interfaz permite arrastrar y colocar (*drag-and-drop*) sobre un escritorio de trabajo los distintos componentes del robot, como sensores y actuadores, establecer las funciones matemáticas y conexiones entre sensores, actuadores y funciones, definir la arquitectura de subsunción, y permite configurar todos estos componentes de manera sencilla. El módulo GUI permite también guardar los comportamientos definidos por el usuario en archivos y abrir los ya almacenados anteriormente para recuperar los comportamientos programados.

Otro de los requerimientos del ERBPI es brindar *independencia de la plataforma.* La aplicación *debe poder utilizarse con una gran variedad de robots y simuladores y ser fácilmente extensible para controlar nuevos robots y simuladores.* Por otra parte, *los usuarios deben ser capaces de probar y ejecutar los mismos comportamientos en múltiples robots de manera sencilla.*

Con este segundo requerimiento, se hace necesaria una capa de abstracción que oculte las particularidades de cada robot y simulador, y exporte una interfaz común al resto de la aplicación. Es decir, que la aplicación trabaje en todo momento con una plataforma robótica abstracta, delegando la conexión y control con cada robot o simulador particular en un módulo dedicado a este propósito (RAL - Robot Abstraction Layer).

Para poder definir esta interfaz es necesario establecer las funciones comunes a todas las plataformas que el ERBPI necesita para poder ejecutar los comportamientos. Para empezar, el ERBPI necesita conocer los sensores y actuadores que posee la plataforma para determinar si es posible ejecutar el comportamiento requerido. Además, necesita conocer los valores de los sensores para calcular, en función de cada comportamiento, los valores para definir a los actuadores. Entonces, requiere cierta funcionalidad que le permita obtener los tipos de sensores y actuadores con los que cuenta el robot, obtener los valores de los sensores y definir nuevos valores para los actuadores. Finalmente, cada plataforma posee un protocolo de conexión y comunicación particular, para lo que también necesita poseer la capacidad de iniciar y finalizar la conexión con el robot adecuadamente.

De aquí se deriva otro requerimiento. Cada plataforma robótica posee sus sensores y actuadores particulares, por lo que cada una maneja valores muy diferentes. Entonces, también se deben abstraer o normalizar los posibles valores distintos de sensores y actuadores que cada plataforma pudiera manejar, por ejemplo, estableciendo un rango homogéneo entre $[0, 100]$ (que puede inter-

pretarse como un porcentaje de activación) para los sensores y un rango entre $[-100, 100]$ (que puede interpretarse como un porcentaje de activación con sentido de giro) para los actuadores.

Por lo tanto, es necesario implementar un módulo RAL que funcione como capa de abstracción e implemente esta interfaz, encargándose de controlar cada plataforma robótica particular. El módulo RAL es el encargado de conocer cómo comunicarse con cada robot o simulador, qué sensores y actuadores realmente posee y manejar homogéneamente sus valores.

Parte del requerimiento es que *la aplicación sea fácilmente extensible para controlar nuevos robots y simuladores*. En este sentido, resulta inconveniente recompilar la aplicación cada vez que surge esta necesidad. Por esto mismo, implementamos el módulo RAL como una librería dinámica que pueda redefinirse en tiempo de ejecución. Además, implementamos archivos de configuración para que el módulo GUI pueda manejar las nuevas plataformas también en tiempo de ejecución.

Hasta el momento, el análisis de requerimientos nos ha permitido identificar claramente dos módulos en la aplicación: los módulos GUI y RAL. Aún falta determinar cómo se interpreta el comportamiento definido gráficamente por el usuario en la interfaz gráfica y quién se encarga de transformarlo en una secuencia de comandos que la plataforma robótica pueda ejecutar. Esta es una responsabilidad que no corresponde a una interfaz gráfica con el usuario ni a un módulo de abstracción de la comunicación y control del robot. Por lo tanto, definimos un tercer módulo de la aplicación, el CORE, para que se encargue de esta tarea.

Así, la aplicación ERBPI queda definida completamente por un conjunto de tres módulos: GUI, CORE y RAL. Una vez definida la responsabilidad de cada módulo, resta establecer una correcta comunicación entre ellos.

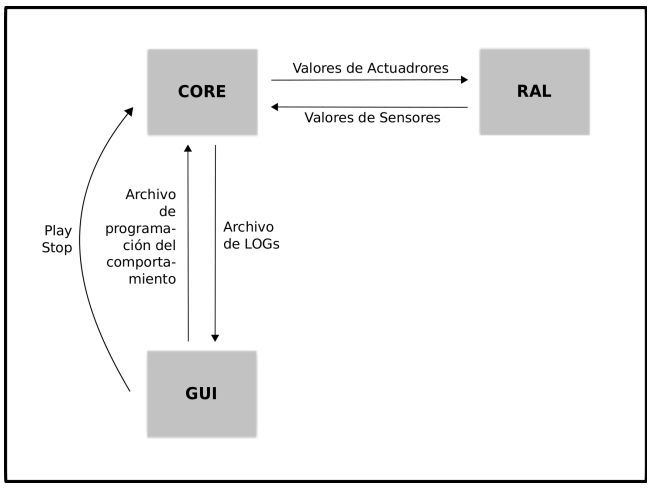

Figura 12: Arquitectura modular del ERBPI.

El CORE se comunica con el RAL para obtener los valores de los sensores y define los valores de los actuadores en función del comportamiento. Para esto,

el CORE se comunica con el RAL, con una interfaz general implementada como librería dinámica, llamando a sus funciones para obtener los valores de sensores y definir los actuadores.

Dado que el GUI provee la capacidad de abrir y guardar los comportamientos definidos mediante archivos, aprovechamos esto para establecer la comunicación entre el módulo GUI y el CORE. A la vez, el CORE lleva un registro de los valores de sensores, actuadores y funciones a cada momento de la ejecución del comportamiento que sirven de comunicación con el GUI.

Por último, otro de los requerimientos del ERBPI es su *portabilidad a diferentes sistemas operativos y plataformas* para ajustarse a los distintos hardwares y softwares disponibles en las escuelas e instituciones educativas. Este requerimiento posee implicancias directas en la implementación, como la utilización de lenguajes de programación como Java, C++ y librerías compilables en distintos sistemas operativos (*crossplatform*), que detallamos en la implementación de cada módulo más adelante.

En resumen, del análisis de requerimientos se desprende que el ERBPI posee un diseño modular que delimita las diferentes responsabilidades de cada módulo en todo el sistema. El módulo GUI (Graphical User Interface) permite al usuario diseñar gráficamente el comportamiento del robot, pudiendo abrir/guardar este comportamiento en un archivo. El módulo CORE es el encargado de leer este archivo, interpretar y ejecutar el comportamiento definido. Para comunicarse con el robot o simulador, el CORE utiliza una capa de abstracción determinada por el módulo RAL (Robot Abstraction Layer). Existe una implementación de RAL para cada plataforma que el ERBPI puede utilizar. El módulo RAL es el encargado de la normalización de todos los valores de sensores y actuadores y, fundamentalmente, el encargado de establecer la comunicación apropiada con el robot mediante su protocolo correspondiente. La arquitectura básica de los tres módulos y la comunicación necesaria entre ellos se muestra en la Figura 12.

A continuación detallamos el diseño e implementación de cada uno de estos módulos.

5.2. Módulo GUI (Graphical User Interface)

Como ya dijimos anteriormente, el módulo GUI es el encargado de proveer la interfaz gráfica del ERBPI con el usuario. A través de este módulo el usuario puede elegir con qué plataforma de robot trabajar (robot real o simulado), programar los distintos comportamientos simples y combinarlos para lograr un comportamiento complejo.

La interfaz gráfica provista por el GUI permite al usuario realizar todas las acciones necesarias para configurar y manejar el ERBPI de forma gráfica, utilizando la técnica de arrastrar y soltar (*drag-and-drop*) sobre el escritorio de trabajo de la interfaz (*work-canvas*) para utilizar los distintos elementos disponibles como comportamientos simples, sensores, funciones, actuadores, transiciones y conexiones.

De esta forma, en el escritorio de trabajo de la interfaz, el usuario puede definir utilizando solamente el mouse los distintos comportamientos simples,

realizar las conexiones entre ellos estableciendo y configurando las distintas transiciones, programar un comportamiento simple seleccionando los sensores, actuadores, estableciendo y configurando las funciones y conectando estos elementos entre sí.

El módulo GUI cumple a la vez la función de ser quien coordina todo el funcionamiento del ERBPI, supervisando la interacción entre los tres módulos. Permite al usuario iniciar la ejecución de los comportamientos y detenerlos presionando los botones correspondientes. El GUI también se encarga de traducir el comportamiento definido en el espacio de trabajo de la interfaz a un archivo de comportamiento e indicarle al módulo CORE con qué archivo de comportamiento y plataforma de robot (RAL) debe trabajar.

La interfaz gráfica del módulo GUI posee dos ventanas principales, una para programar los comportamientos simples y otra para el comportamiento complejo, que detallaremos a continuación.

5.2.1. Comportamiento Simple

Basado en el modelo de los Vehículos de Braitenberg, la ventana de programación del comportamiento simple brinda las herramientas necesarias para establecer las distintas conexiones entre los sensores, funciones y actuadores del robot.

Esta ventana presenta un esquema del robot donde todos los sensores del mismo se encuentran en el centro. A la izquierda y la derecha de los sensores se encuentran los actuadores, la rueda izquierda y rueda derecha del robot. Entre los sensores y actuadores, el escritorio de trabajo posee una zona donde pueden realizarse las conexiones y colocarse las funciones seleccionadas de un conjunto predefinido.

En las Figuras 13, 14 y 15 puede verse un esquema de conexión para el robot ExaBot. Las conexiones pueden establecerse con el mouse presionando en el elemento origen de la conexión y arrastrando hasta el elemento destino. Las funciones también pueden arrastrarse y soltarse directamente en el escritorio de trabajo con el mouse.

La ventana brinda una barra de herramientas donde se encuentran las funciones predefinidas. Entre ellas se encuentran la función *excitatoria* o lineal positiva, la función *inhibitoria* o lineal negativa, la función *partida* parametrizable y la función *constante* también parametrizable. En la Figura 16 se describe el esquema de estas funciones. A la función *constante* no se le pueden definir entradas.

Las funciones parametrizables, como la *partida* o la *constante*, pueden configurarse mediante una ventana emergente que proporciona el menú contextual de cada elemento. En el caso de la función *partida* se realiza mediante la composición de los dos puntos que la definen, colocándolos con el mouse en la posición deseada. Lo mismo sucede con la función *constante* pero sólo con un valor que define el resultado constante de la función. En la Figura 17 se describen el menú contextual de los elementos y las ventanas emergentes de configuración de parámetros de las funciones.

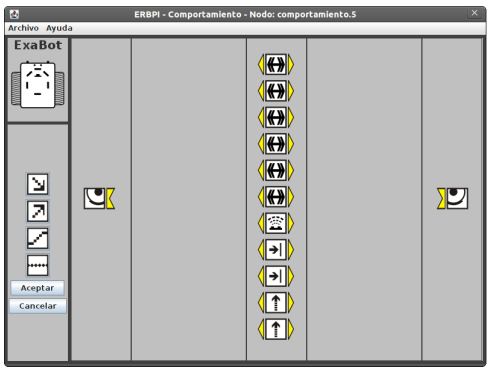

Figura 13: Ventana del escritorio de trabajo inicial para la programación de un comportamiento simple con el robot ExaBot.

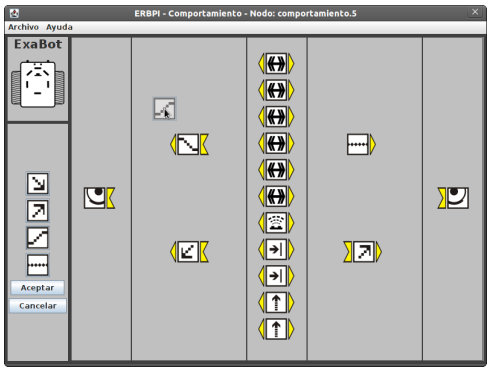

Figura 14: Arrastrando y soltando una función sobre el escritorio de trabajo para la programación de un comportamiento simple con el robot ExaBot.

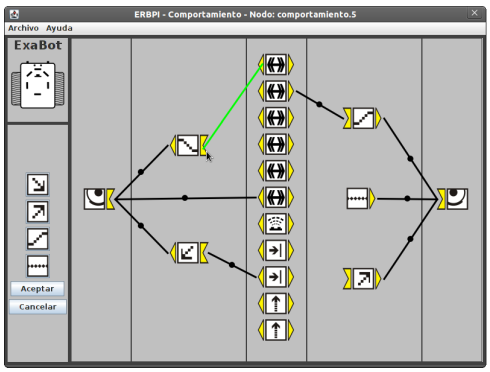

Figura 15: Estableciendo conexiones entre los distintos elementos para la programación de un comportamiento simple con el robot ExaBot.

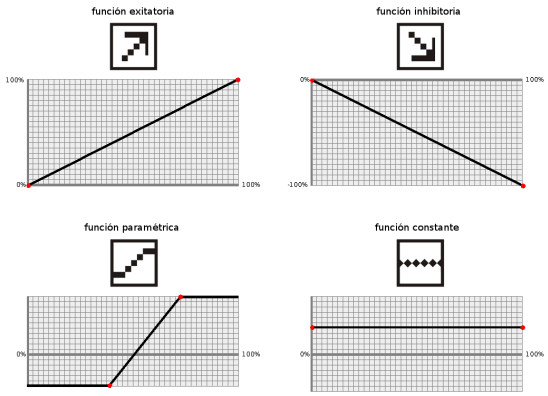

Figura 16: Esquema de funciones utilizadas en un comportamiento simple.

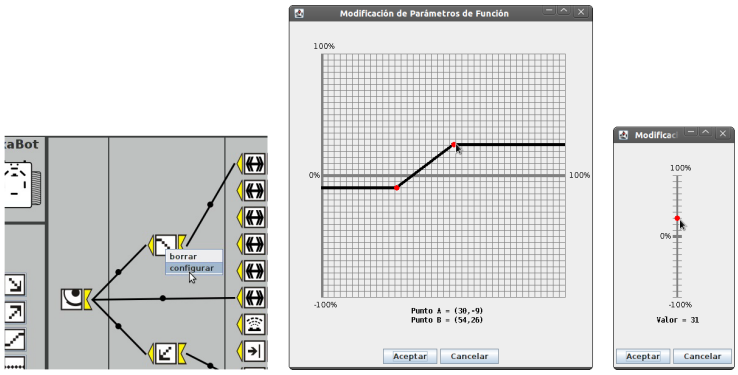

Figura 17: Configuración de parámetros de funciones.

Los sensores que actualmente maneja el ERBPI son sensores infrarrojos de proximidad, sonar, de luz, de contacto y de detección de línea. En la Figura 18 pueden verse los distintos esquemas utilizados para cada sensor.

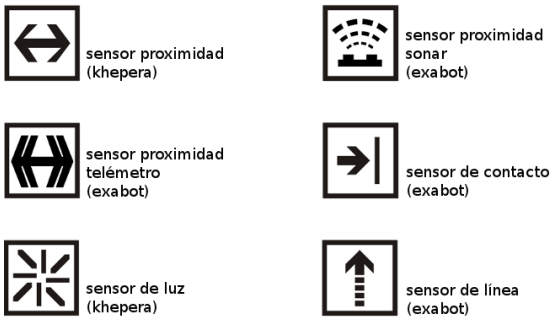

Figura 18: Esquema de los sensores utilizados para la programación de comportamientos simples.

Además, esta ventana muestra un esquema del robot que permite relacionar sus sensores y actuadores con los que se están utilizando en el escritorio de trabajo. Cuando el puntero del mouse se posa sobre algún sensor o actuador en el escritorio de trabajo, éste es coloreado automáticamente en el esquema del robot, lo que permite visualizar rápidamente qué sensores son los que están conectados y cómo inciden en el comportamiento simple que se está programando. En la Figura 19 puede verse un ejemplo del robot ExaBot y esta funcionalidad.

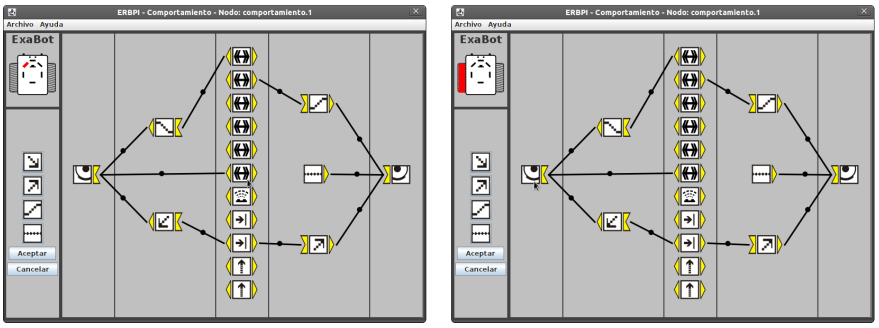

Figura 19: Visualización de la correlación de sensores y actuadores en el escritorio de trabajo con el robot. En el primer ejemplo el mouse se encuentra sobre uno de los sensores infrarrojos de proximidad y éste es marcado en color rojo en el esquema del robot. En el segundo caso, sucede los mismo pero con el actuador izquierdo del robot.

Por último, esta ventana también brinda la posibilidad de abrir y guardar comportamientos simples. De esta forma, pueden reutilizarse comportamientos simples ya programados con anterioridad.

Una vez finalizada la programación del comportamiento simple, el usuario puede presionar el botón *Aceptar* para establecer finalmente este comportamiento o el botón *Cancelar* para deshacer los cambios. En ambos casos se retorna a la ventana de programación del comportamiento complejo.

5.2.2. Comportamiento Complejo

Mediante una arquitectura de subsunción basada en comportamientos, la ventana de programación del comportamiento complejo brinda las herramientas necesarias para combinar comportamientos simples estableciendo transiciones y condiciones entre ellos. En adelante, llamaremos también *nodos* a los comportamientos simples de acuerdo a la nomenclatura utilizada en las arquitecturas de subsunción.

La interfaz de programación del comportamiento complejo del GUI es al mismo tiempo la ventana principal de la interfaz. Permite seleccionar con qué plataforma de robot trabajar, programar los comportamientos simples o nodos, establecer contadores y temporizadores globales y realizar las combinaciones de los comportamientos simples determinando un comportamiento complejo. A la vez esta ventana es la que permite al usuario comenzar y detener la ejecución del comportamiento en el robot o simulador.

Para comenzar a utilizar la interfaz, el usuario crea un nuevo comportamiento complejo seleccionando del menú con cuál robot o simulador trabajar. Luego puede comenzar a definir los comportamientos simples. Para esto, presiona con el botón del mouse sobre el escritorio de trabajo colocando un nuevo nodo. Para programar este nodo, puede hacer doble click con el mouse sobre el icono del nodo, lo que abre una ventana de programación de comportamiento simple como se detalla en el punto anterior. Para establecer una nueva transición entre dos comportamientos simples, el usuario hace click sobre el nodo origen de la transición y arrastra el puntero hacia el nodo destino.

En las Figuras 20, 21 y 22 puede verse la programación del comportamiento complejo combinando distintos comportamientos simples.

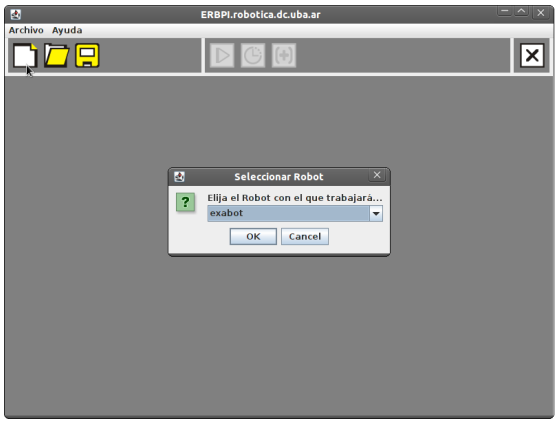

Figura 20: Ventana del escritorio de trabajo inicial donde se selecciona crear un nuevo comportamiento complejo para el robot ExaBot.

El usuario puede definir una serie de temporizadores y contadores para configurar las transiciones mediante botones de la barra de menú y ventanas emergentes. En la Figura 23 pueden verse estas ventanas.

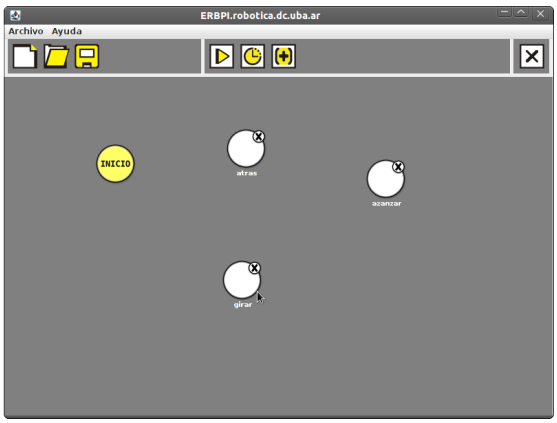

Figura 21: Estableciendo nuevos comportamientos simples en el escritorio de trabajo para definir un comportamiento complejo para el robot ExaBot.

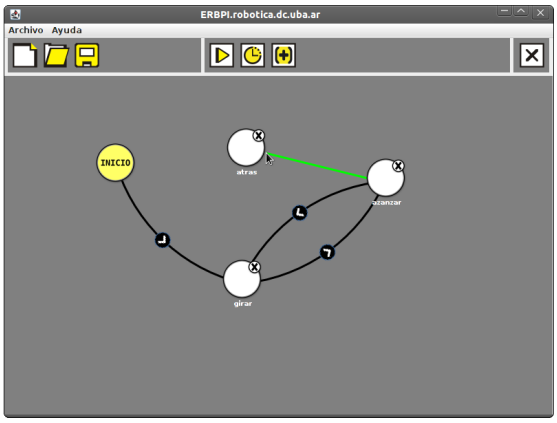

Figura 22: Estableciendo transiciones entre los nodos cuya combinación determina un comportamiento complejo para el robot ExaBot.

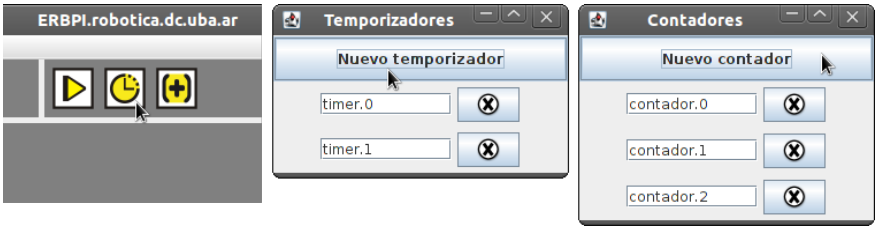

Figura 23: Configuración del conjunto global de contadores y temporizadores. De izquierda a derecha, la barra de menú con los botones para agregar temporizadores y contadores; la ventana emergente para definir nuevos temporizadores; y la ventana emergente para definir nuevos contadores.

Las transiciones pueden configurarse estableciendo condiciones sobre las mismas. Cuando un comportamiento simple origen de una transición se está ejecutando y las condiciones de esa transición se cumplen, la ejecución pasa al comportamiento simple destino. Las condiciones se configuran con una ventana emergente que proporciona el menú contextual de cada transición. Esta ventana emergente permite trabajar con operadores lógicos predefinidos ($<$, $\leq$, $=$, $>$ y $\geq$) y los valores de sensores del robot, contadores y temporizadores para definir las condiciones de la transición.

Al mismo tiempo, el usuario puede definir actualizaciones sobre los temporizadores y contadores que se ejecutarán al realizarse la transición. Los contadores pueden *resetearse*, *incrementarse* y *decrementarse*, mientras que los temporizadores sólo pueden *resetearse*.

En la Figura 24 se describen el menú contextual de las transiciones y la ventana emergente de configuración de las condiciones.

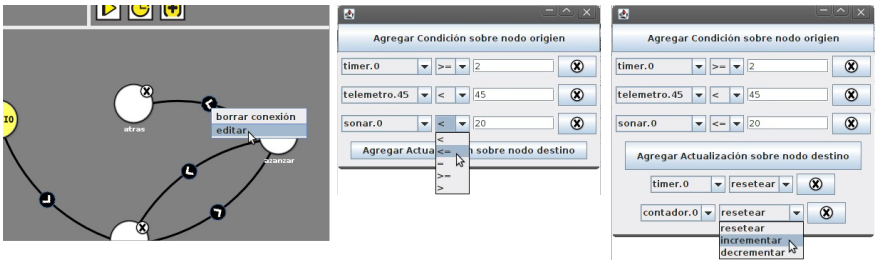

Figura 24: Configuración de condiciones y actualizaciones de transiciones.

En el caso de existir varias transiciones de salida de un nodo que se está ejecutando, cuando las condiciones de más de una se cumplan simultáneamente, se selecciona determinísticamente una de ellas.

Finalmente, es necesario definir cuál de todos los nodos es el que se ejecutará primero. Para esto, el usuario debe definir una transición de salida desde el *INICIO*, identificado como tal y en otro color en el escritorio de trabajo, hacia el nodo o comportamiento simple que se ejecutará primero. El *INICIO* y su respectiva transición de salida son simplemente un recurso gráfico para indicar cuál será el comportamiento simple que se ejecutará primero. Por lo tanto, el *INICIO* no representa un comportamiento y, al igual que a su transición de salida, no se le pueden realizar configuraciones de ningún tipo.

Además, la interfaz brinda la posibilidad de agregar descripciones a cada nodo para mayor declarativiad del comportamiento programado. Para definir las descripciones de cada nodo, el usuario puede acceder a una ventana emergente a través del menú contextual de cada nodo.

En la Figura 25 se describen el menú contextual de los nodos, la ventana emergente de configuración de las descripciones y el *INICIO*.

Una vez finalizado el comportamiento complejo, el usuario puede ejecutar este comportamiento presionando el botón de ejecutar o *play* que se encuentra en la barra de menú principal. El GUI transforma el comportamiento definido gráficamente en un archivo temporal de comportamiento para que el módulo

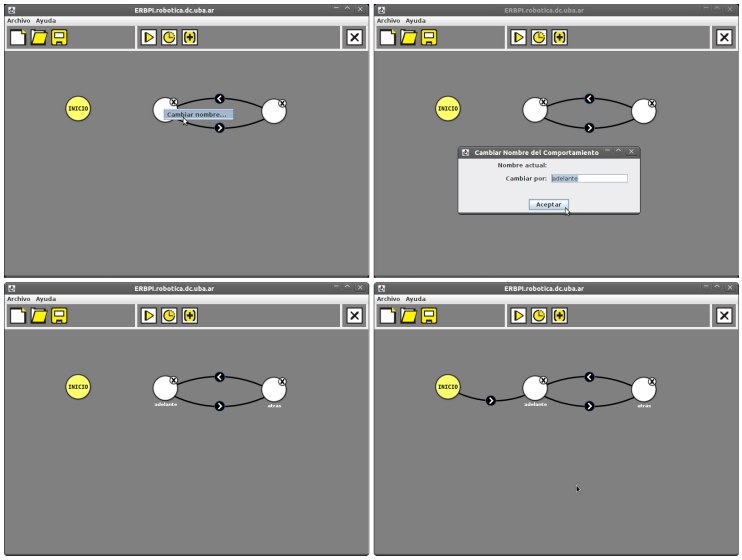

Figura 25: Configuración del INICIO y de las descripciones de los comportamientos simples. De izquierda a derecha, dos nodos determinando un comportamiento complejo y el menú contextual para definir la descripción del nodo; ventana emergente para definir la descripción; el primer nodo es descripto como *adelante* y el segundo como *atrás*; el nodo *adelante* es definido como el nodo que primero se ejecutará tras definirlo como nodo destino de la transición de salida del *INICIO*.

CORE lo pueda ejecutar. Al comenzar la ejecución, el GUI muestra una ventana emergente donde indica el comienzo de la ejecución del comportamiento. Para detener la ejecución el usuario presiona el botón *parar* en esta ventana emergente. Adicionalmente, esta ventana emergente muestra además algunos mensajes que podrían ser útiles para el usuario, como por ejemplo, que el comportamiento ha podido ser interpretado correctamente y se comenzará la ejecución, cuál es el comportamiento simple que se está ejecutando actualmente y mensajes de error. En la Figura 26 puede verse un ejemplo de ejecución para el simulador ExaSim.

Por último, esta interfaz brinda la posibilidad de guardar y abrir comportamientos complejos, mediante la barra de menú principal.

5.2.3. Implementación

El módulo GUI está implementado en el lenguaje de programación Java. Elegimos este lenguaje de programación ya que es un buen lenguaje para interfaces gráficas y también por su portabilidad a distintos sistemas operativos sin la necesidad de tener que recompilarlo. El único requerimiento para su ejecución es la instalación de JVM (Java Virtual Machine).

La implementación cuenta de dos capas. Por un lado, la capa de bajo nivel que denominamos *model*. Esta capa es la que agrupa las distintas clases y objetos que le dan toda la funcionalidad al GUI. Por el otro, un capa de alto nivel que

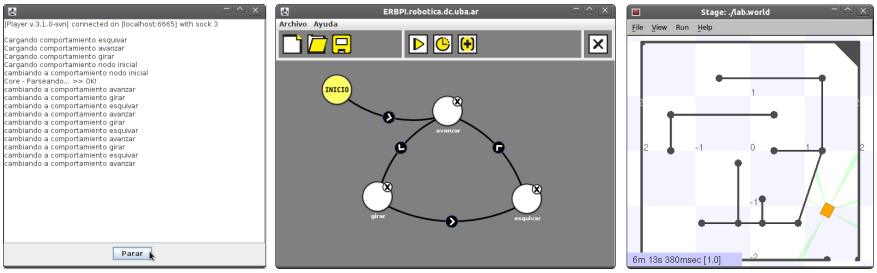

Figura 26: Ejemplo de ejecución de un comportamiento complejo con el simulador ExaSim. Pueden verse simultáneamente durante la ejecución la ventana de la interfaz gráfica donde se encuentra definido el comportamiento complejo que se ejecutó mediante el botón *play*; la ventana emergente de ejecución; y la ventana donde se está ejecutando el comportamiento en el simulador ExaSim.

denominamos *gui* propiamente dicho. Esta capa es la que agrupa todas las clases y objetos necesarios para brindar la interfaz gráfica y la interacción con el usuario.

Las configuraciones relevantes del GUI se definen a través de archivos de configuración sin la necesidad de modificar o recompilar el módulo. Se tienen dos archivos de configuración, uno para características del GUI y otro para las características de las plataformas de robots:

- **Configuración del módulo:** Este archivo de configuración brinda al GUI la posibilidad de modificar y extender sus características en cualquier momento. Utiliza formato XML (Extensible Markup Language) [46]. Entre las características más importantes que pueden modificarse o extenderse se encuentran los tipos de sensores y actuadores que el GUI será capaz de manejar . También la forma en que estos objetos se visualizan gráficamente en la interfaz gráfica y cuáles son las imágenes asociadas a los mismos. Otro elemento importante de la configuración es el conjunto de funciones predefinidas que el GUI brinda para la programación de los comportamientos simples (actualmente las funciones *excitatoria*, *inhibitoria*, *partida* y *constante*). También establece cómo se visualizan gráficamente, y cuáles son los valores por defecto de los parámetros de las funciones.

- **Configuración de los robots:** Este archivo de configuración brinda al GUI la posibilidad de conocer y poder manejar cada plataforma robótica correctamente. Al igual que el archivo de configuración del GUI, utiliza el formato XML. Brinda la posibilidad de agregar nuevos robots, modificar y extender las características de los mismos. Cada plataforma de robot posee su propio archivo de configuración, (actualmente contamos con *khepera.xml*, *yaks.xml*, *exabot.xml* y *exasim.xml*). Este archivo indica cuál es el módulo RAL correspondiente al robot, si el robot posee la capacidad de procesamiento para ejecutar el comportamiento directamente en el robot, qué sensores y actuadores posee, la cantidad y características de cada sensor y actuador, la ubicación de cada uno en el cuerpo robot, cómo se visualizan gráficamente los mismos en el GUI y el esquema o representación gráfica de todo el robot.

La combinación de estos archivos de configuración del GUI y de los robots es una de las características más importantes de su implementación. Así, resulta rápido y sencillo modificar o extender las características del GUI para incorporar nuevas funcionalidades y plataformas de robots. Por ejemplo, en el caso de tener un nuevo robot con un nuevo tipo de sensor que hasta el momento el ERBPI no maneja, simplemente se requiere agregar las características del nuevo sensor en el archivo de configuración del GUI y agregar un nuevo archivo de configuración para el nuevo robot. Luego se reinicia la interfaz gráfica y automáticamente el ERBPI es capaz de utilizar el nuevo robot y el nuevo tipo de sensor para la programación de comportamientos.

Pueden verse dos ejemplos simples del archivo de configuración del GUI y de los robots en los Cuadros 1 y 2.

```
<configuracionGUI>
    <tipoSensores>
        <tipoSensor id='proximidad'>
            <imagen href='images/sen_proximidad.png'/>
        </tipoSensor>
        <tipoSensor id='sonar'>
            <imagen href='images/sen_sonar.png'/>
        </tipoSensor>
    </tipoSensores>
    <tipoActuadores>
        <tipoActuador id='rueda'>
            <imagen href='images/act_rueda.png'/>
        </tipoActuador>
    </tipoActuadores>
    <herramientas>
        <esquemaFuncion id='exitatoria' aceptaEntradas='true'
             parametrizable='false'>
            <imagen href='images/f_exitatoria.png'/>
            <puntos>
                <punto x='0' y='0'/>
                <punto x='100' y='100'/>
            </puntos>
        </esquemaFuncion>
        <esquemaFuncion id='partida' aceptaEntradas='true'
             parametrizable='true'>
            <imagen href='images/f_partida.png'/>
            <puntos>
                <punto x='25' y='0'/>
                <punto x='75' y='0'/>
            </puntos>
        </esquemaFuncion>
        <esquemaFuncion id='constante' aceptaEntradas='false'
             parametrizable='true'>
            <imagen href='images/f_constante.png'/>
            <puntos>
                <punto x='0' y='20'/>
                <punto x='100' y='20'/>
            </puntos>
        </esquemaFuncion>
    </herramientas>
</configuracionGUI>
```

Cuadro 1: Ejemplo simple de archivo de configuración del GUI.

El comportamiento complejo, definido gráficamente en la interfaz, es transformado por el GUI a un archivo de comportamiento implementado también en formato XML. Este archivo de comportamiento contiene todos los datos necesarios para que el CORE pueda llevar adelante la ejecución del comportamiento.

```
<robot id="exabot" nombre="ExaBot" ejecucionLocal='true'>
    <imagen id='esquema' href='robots/robot_exabot.png' />
    <ral href='libexabotRAL.so'/>
    <sensores>
        <sensor id='telemetro.45' tipo='telemetro' default='true'>
            <mapaimagen imagen='esquema'>
                <linea x0='0.63' y0='0.30' x1='0.68' y1='0.35'
                    width='5' color='255,0,0'/>
            </mapaimagen>
        </sensor>
        (...)
        <sensor id='sonar.0' tipo='sonar' default='true'>
            <mapaimagen imagen='esquema'>
                <linea x0='0.49' y0='0.36' x1='0.55' y1='0.36'
                    width='5' color='255,0,0'/>
            </mapaimagen>
        </sensor>
        (...)
        <sensor id='contacto.0' tipo='contacto' default='true'>
            <mapaimagen imagen='esquema'>
                <linea x0='0.33' y0='0.11' x1='0.36' y1='0.11'
                    width='3' color='255,0,0'/>
            </mapaimagen>
        </sensor>
        (...)
    </sensores>
    <actuadores>
        <actuador id='rueda.izquierda' nombre='rueda izq' tipo='rueda'
             default='true'>
            <mapaimagen imagen='esquema'>
                <linea x0='0.20' y0='0.40' x1='0.20' y1='0.83'
                    width='12' color='255,0,0'/>
            </mapaimagen>
            <ubicacion id='izquierda'/>
        </actuador>
        (...)
    </actuadores>
</robot>
```

Cuadro 2: Ejemplo simple de archivo de configuración del robot ExaBot.

A la vez, el archivo de comportamiento sigue el estándar DOM (Document Object Model) [47], permitiendo establecer fácilmente la jerarquía entre los distintos elementos para la ejecución y definir los objetos necesarios con sus respectivos atributos, propiedades y funciones.

Esta estructura jerárquica tiene como primer objeto el *comportamientoComplejo*. De este objeto se desprenden el resto:

- `robot`: indica en qué robot será ejecutado el comportamiento.
- `sensores`: indica el conjunto de sensores utilizados. A la vez, establece el conjunto mínimo de sensores que debe poseer el robot para poder llevar adelante el comportamiento.
- `actuadores`: indica el conjunto de actuadores utilizados. A la vez, establece el conjunto mínimo de actuadores que debe poseer el robot para poder llevar adelante el comportamiento.
- `timers`: indica el conjunto de temporizadores globales.
- `contadores`: indica el conjunto de contadores globales.

- `comportamientoSimple`: indica la definición de un comportamiento simple.
- `transiciones`: indica la distintas transiciones, condiciones y actualizaciones entre los comportamientos simples.

A la vez, cada *comportamientoSimple* posee también una cantidad objetos que se desprenden de él:

- `funciones`: indica el conjunto de funciones utilizadas.
- `conexiones`: indica el conjunto de conexiones definidas entre los distintos sensores, funciones y actuadores.

Puede verse un ejemplo simple del archivo de comportamiento en el Cuadro 3.

Este ejemplo consiste en un comportamiento complejo definido a partir de dos comportamientos simples *buscarLinea* y *seguirLinea*. El comportamiento comienza por el nodo *buscarLinea* que hace que el robot gire en el lugar hasta detectar la línea en el piso con el sensor de línea *linea.0*. En ese caso se cumple la condición de la transición y la ejecución pasa al nodo *seguirLinea* que utiliza los dos sensores de línea para avanzar siguiendo la línea en el piso.

Por último, en la Figura 27 puede verse la definición de este comportamiento mediante la interfaz gráfica del GUI.

5.3. Módulo CORE

El módulo CORE es el encargado de ejecutar el comportamiento. Para esto, debe leer el archivo donde está almacenado y establecer la conexión con el módulo RAL.

La comunicación básica entre el CORE y el RAL consiste en que el CORE recibe del RAL los valores normalizados de los sensores, ejecuta el comportamiento y envía al RAL los nuevos valores normalizados para los actuadores. El CORE finaliza su ejecución cuando el GUI envía la señal de detenerse a partir de la solicitud del usuario.

Para poder ejecutar cada comportamiento simple, el CORE debe transformarlo en una secuencia ejecución. Para esto el CORE interpreta un comportamiento simple como un grafo de ejecución, donde los nodos son los sensores, funciones y actuadores y las aristas las conexiones entre ellos. Luego, el CORE establece un orden topológico [48] del grafo de ejecución asociado al comportamiento, garantizando que las entradas de cada función y actuador hayan sido calculadas con anterioridad para cuando sean necesarias durante la ejecución. De esta forma, el CORE infiere un programa imperativo equivalente al comportamiento simple asociado.

El CORE también realiza los chequeos necesarios para garantizar que el comportamiento definido por el usuario pueda ser ejecutado en el robot seleccionado. Esto significa que el comportamiento definido por el usuario da lugar a un

```
<comportamientoComplejo id_comportamientoSimpleInicial="comportamientoSimple.1">
    <robot id="exabot"/>
    <sensores>
        <sensor id="linea.0"/>
        <sensor id="linea.1"/>
    </sensores>
    <actuadores>
        <actuador id="rueda.izquierda"/>
        <actuador id="rueda.derecha"/>
    </actuadores>
    <timers>
        <timer descripcion="timer.0" id="timer.0"/>
    </timers>
    <contadores>
        <contador descripcion="contador.0" id="contador.0"/>
    </contadores>
    <comportamientoSimple descripcion="buscarLinea" id="comportamientoSimple.1">
        <funciones>
            <funcion id="funcion.4" location="left" templateId="constante">
                <punto x="0" y="20"/>
                <punto x="100" y="20"/>
            </funcion>
            <funcion id="funcion.5" location="right" templateId="constante">
                <punto x="0" y="-20"/>
                <punto x="100" y="-20"/>
            </funcion>
        </funciones>
        <conexiones>
            <conexion dst="rueda.izquierda" src="funcion.4"/>
            <conexion dst="rueda.derecha" src="funcion.5"/>
        </conexiones>
    </comportamientoSimple>
    <comportamientoSimple descripcion="seguirLinea" id="comportamientoSimple.2">
        <funciones>
            <funcion id="funcion.0" location="left" templateId="exitatoria">
                <punto x="0" y="0"/>
                <punto x="100" y="100"/>
            </funcion>
            <funcion id="funcion.1" location="right" templateId="exitatoria">
                <punto x="0" y="0"/>
                <punto x="100" y="100"/>
            </funcion>
            <funcion id="funcion.2" location="left" templateId="constante">
                <punto x="0" y="20"/>
                <punto x="100" y="20"/>
            </funcion>
            <funcion id="funcion.3" location="right" templateId="constante">
                <punto x="0" y="20"/>
                <punto x="100" y="20"/>
            </funcion>
        </funciones>
        <conexiones>
            <conexion dst="rueda.derecha" src="funcion.1"/>
            <conexion dst="rueda.derecha" src="funcion.3"/>
            <conexion dst="rueda.izquierda" src="funcion.0"/>
            <conexion dst="rueda.izquierda" src="funcion.2"/>
            <conexion dst="funcion.1" src="linea.0"/>
            <conexion dst="funcion.0" src="linea.1"/>
        </conexiones>
    </comportamientoSimple>
    <transiciones>
        <transicion id_destino="comportamientoSimple.2"
             id_origen="comportamientoSimple.1">
            <condiciones>
                <condicion comparacion=">=" id_elemento="linea.0" umbral="100"/>
            </condiciones>
            <actualizaciones>
                <actualizacion accion="incrementar" id_contador="contador.0"
                   tipo="contador"/>
                <actualizacion accion="resetear" id_timer="timer.0" tipo="timer"/>
            </actualizaciones>
        </transicion>
    </transiciones>
</comportamientoComplejo>
```

Cuadro 3: Ejemplo simple de archivo de comportamiento generado por el GUI. La Figura 27 muestra la correlación de este comportamiento con el esquema definido gráficamente en el GUI.

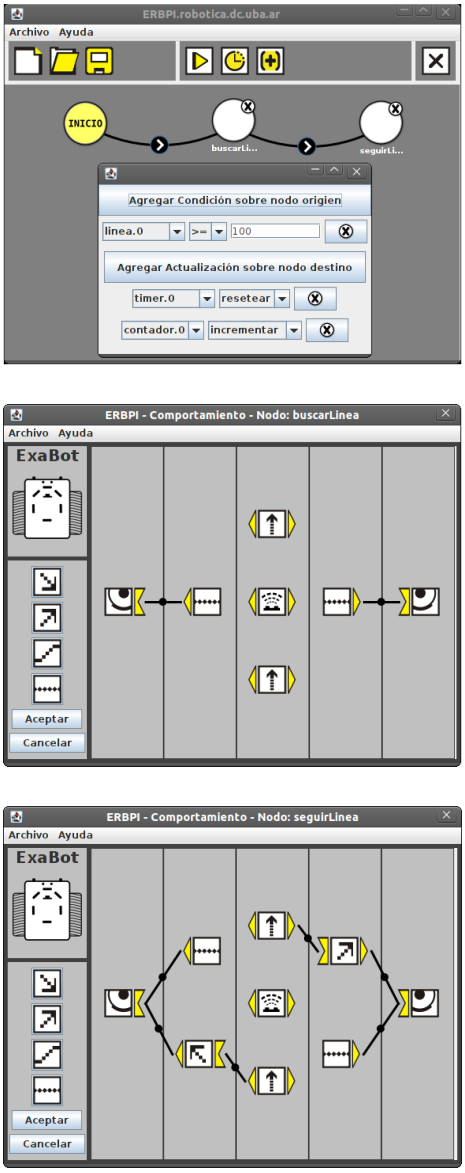

Figura 27: Esquema de la correlación del archivo de comportamiento definido en el Cuadro 3 con el visualizado gráficamente en el GUI. Puede verse el comportamiento complejo y las condiciones en la transición y los esquemas definidos para los comportamientos simples *buscarLinea* y *seguirLinea*.

grafo acíclico que pueda ser ejecutado secuencialmente y que el robot dispone de los sensores y actuadores suficientes para ejecutar el comportamiento. El CORE establece la frecuencia de trabajo y comunicación con el RAL (medida en ciclos de control por segundo), dado que cada robot y simulador tiene una frecuencia de trabajo diferente dependiendo de sus unidades de procesamiento, sensores, actuadores y protocolos de comunicación. El CORE también mantiene un archivo log donde registra los valores de cada sensor, función y actuador en cada momento.

Otra característica importante es que, dependiendo de la capacidad de procesamiento del robot, el CORE puede ejecutarse directamente en el hardware embebido del robot en lugar de hacerlo remotamente. Esto disminuye considerablemente la comunicación entre el robot y el ERBPI, aumentando el desempeño global de la aplicación y la experiencia del usuario. Este es el caso por ejemplo del robot ExaBot, que tiene la capacidad de ejecutar programas localmente porque tiene un procesador embebido.

5.3.1. Implementación

El CORE está implementado en el lenguaje de programación C++. Elegimos este lenguaje de programación debido a que el CORE es el encargado de la ejecución del comportamiento en los robots y, por lo tanto, es necesario que esta implementación sea lo más eficiente posible y cercana a la ejecución en tiempo real.

El CORE puede ser ejecutado localmente en los robots cuando la capacidad de cómputo de los mismos lo permitan. De esta manera, se elimina la necesidad de transmisión de cada uno de los comandos, logrando que la ejecución del comportamiento sea más rápida y estableciendo un comportamiento realmente reactivo. Para esto, el CORE también se encuentra modularizado. La funcionalidad básica y común a cualquier implementación está agrupada en una librería que denominamos *libcore*.

La librería *libcore* exporta las siguientes funciones: *core-initialize*, *core-start*, *core-execute*, *core-stop* y *core-deinitialize*.

El CORE propiamente dicho, es quien utiliza la librería *libcore*. Actualmente existen dos implementaciones del CORE, una para la ejecución remota en cualquier plataforma robótica que no tenga capacidad de cómputo para ejecución local, y otra para la ejecución local en el robot ExaBot.

Esta modularización del CORE permite extender fácilmente al ERBPI para contemplar el agregado de futuros robots que posean la capacidad de cómputo suficiente para ejecutar el CORE localmente.

A continuación, describimos el funcionamiento del CORE y cada una de sus partes:

5.3.2. CORE

Como detallamos antes, el CORE se reduce a un programa simple mediante la utilización de la librería *libcore*.

Podemos describir el pseudocódigo general del CORE de la siguiente forma:

1. Llamar a las funciones *core-initialize* y *core-start* de la *libcore* para establecer la conexión con el RAL y transformar el archivo de comportamiento generando las estructuras y objetos necesarios para la ejecución.
2. Repetir la llamada a la función *core-execute* de la *libcore* para ejecutar el comportamiento hasta recibir la señal de terminar.
3. Llamar a las funciones *core-stop* y *core-deinitialize* de la *libcore* para detener el robot adecuadamente y finalizar la conexión con el RAL.

Como el CORE entra en un ciclo infinito es necesario implementar la atención de señales de sistema. Para terminar la ejecución, el CORE tiene definidas las rutinas de atención para las señales de finalización (SIGTERM) e interrupción (SIGINT). El manejo y atención de señales permite, por un lado, manejar correctamente la finalización del CORE y el RAL y, por consiguiente, detener el robot de forma apropiada evitando posibles daños en el mismo. Por otro lado, otorga al GUI la capacidad de controlar el inicio y finalización de la ejecución del CORE y del robot.

Por último, si bien es esperable que el CORE sea ejecutado por el GUI, el diseño modular del ERBPI le permite al CORE ser un módulo autónomo. De esta forma, puede ejecutar comportamientos en los robots indicándole como parámetros los archivos de comportamiento y de log.

5.3.3. libcore

La librería *libcore* agrupa la funcionalidad básica y común a todas las implementaciones del CORE.

Esta librería proporciona la siguiente funcionalidad:

core-start:

Esta función se encarga del preprocesamiento necesario para poder comenzar la ejecución del comportamiento. Su pseudocódigo es el siguiente:

1. Parsear el comportamiento: leer el archivo de comportamiento y realizar los chequeos necesarios de consistencia y factibilidad de ejecución del mismo.
2. Generar estructuras y objetos necesarios para la ejecución del comportamiento.

3. Realizar un chequeo de suficiencia de sensores y actuadores: chequear que el robot contiene el conjunto de sensores y actuadores utilizados en el comportamiento para su ejecución, proporcionados por el RAL.
4. Definir la frecuencia de trabajo: obtener la frecuencia de trabajo apropiada proporcionada por el RAL para luego no exceder esta frecuencia en la ejecución.
5. Establecer según el archivo de comportamiento cuál es el comportamiento simple inicial a ejecutar.

core-execute:

Esta función se encarga de actualizar el valor de los actuadores a partir del valor de los sensores y del comportamiento definido. Su pseudocódigo es el siguiente:

1. Obtener del RAL los nuevos valores de sensores en este momento.
2. Actualizar las estructuras y objetos necesarios con estos valores.
3. Si se cumple la condición de alguna transición de salida para el comportamiento simple actual:
   a) Realizar las actualizaciones necesarias de esa transición.
   b) Cambiar el comportamiento simple actual al nuevo comportamiento simple destino de la transición.
4. Ejecutar el comportamiento simple actual:
   a) En función de los nuevos valores de sensores, calcular los nuevos valores de salida de cada función.
   b) En función de las entradas definidas para cada actuador y los nuevos valores de sensores y de salida de funciones, calcular los nuevos valores de los actuadores.
5. Enviar al RAL los nuevos valores de actuadores en este momento.
6. Actualizar los contadores y temporizadores globales.
7. Actualizar el log con los valores de todos los elementos del comportamiento en este momento.
8. Esperar el tiempo necesario para cumplir con la frecuencia de trabajo del robot.

core-stop:

Esta función se encarga de detener el robot adecuadamente, enviándole los comandos adecuados a los actuadores.

Diagrama de clases

El CORE está implementado mediante una jerarquía de clases en C++ siguiendo la programación orientada a objetos. En la Figura 28 puede verse el diagrama de clases en UML (Unified Modeling Language).

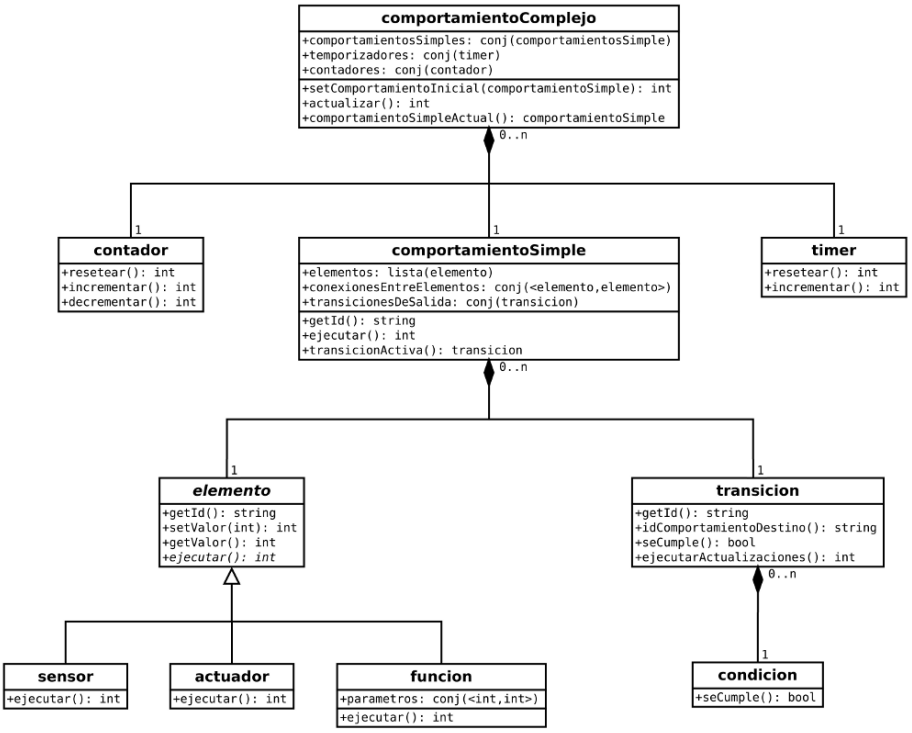

Figura 28: Diagrama de clases del CORE.

Para maximizar la performance de la ejecución del CORE y obtener un código más fácilmente mantenible, se utilizan las clases, estructuras y tipos de datos provistos por la librería STL (Standard Template Library) de C++ [49] como *string*, *vector*, *set* y *map*.

El comportamiento simple queda determinado por una lista ordenada de elementos (sensores, funciones y actuadores) y por un conjunto de conexiones entre ellos que definen el orden topológico y, por tanto, el orden de la lista de elementos. Así, ejecutar el comportamiento consiste simplemente en recorrer secuencialmente esta lista, aplicando el método *ejecutar* de cada elemento.

El comportamiento complejo queda determinado por un conjunto de comportamientos simples, temporizadores y contadores. Así, una vez determinado el comportamiento simple actual, ejecutar el comportamiento complejo consiste simplemente en ejecutar el método *actualizar*.

Parser

El primer paso de la función *core-start* es parsear el archivo de comportamiento.

El archivo de comportamiento contiene información adicional, que el parser del CORE ignora, pero que sirven a otros módulos como el GUI. Esto permite poder extender con facilidad distintas funcionalidades a través del archivo de comportamiento, tanto para el módulo CORE como para el módulo GUI, sin que las funcionalidades de uno afecten al otro.

El parser recorre el archivo de comportamiento, buscando las etiquetas *comportamientoComplejo*, *comportamientoSimple*, *sensor*, *actuador*, *funcion*, *transicion*, *timer*, *contador* y construye e instancia los objetos vistos en el Punto 5.3.3.

En el Cuadro 3 del Punto 5.2.3 puede verse un detalle del archivo de comportamiento en formato XML que el Parser se encarga de interpretar.

5.3.4. Dependencias

Para que el CORE sea portable a distintos sistemas operativos, se utilizaron algunas librerías externas de C++ compilables en distintos sistemas operativos. Estas librerías *crossplatform* son *Xerces-C++ XML Parser* para realizar el parseo del archivo de comportamiento y *Boost C++ System* para la comunicación con el sistema operativo. Estas librerías son de fácil obtención e instalación.

5.4. Módulo RAL (Robot Abstraction Layer)

El módulo RAL es el encargado de encapsular todo el conocimiento particular del robot o simulador. De esta forma, proporciona una interfaz estándar para el módulo CORE. El RAL es quien maneja todo lo necesario para comunicarse con los robots.

El módulo RAL proporciona una capa de abstracción respecto del hardware o software específico que se requiere controlar, es decir, por ejemplo, qué tipo de robot o simulador, qué tipo y cantidad de sensores y actuadores posee el robot y qué tipo de protocolo de comunicación se requiere para la conexión. Permite al módulo CORE obtener la lista de sensores y actuadores del robot, la frecuencia de trabajo del robot y poder trabajar con valores normalizados para los sensores y actuadores. Luego, será el módulo RAL el que se comunicará directamente con el robot o simulador según corresponda.

Para agregar un nuevo robot o simulador al ERBPI, el usuario o programador, simplemente debe programar un RAL específico para ese robot, respetando la interfaz general que brinda el RAL, y agregar las configuraciones necesarias en los archivos de configuración del GUI mencionados anteriormente en el Punto 5.2.3.

5.4.1. Implementación

El RAL para cada robot o simulador está implementado como una librería dinámica del sistema operativo. De esta forma, se pueden agregar nuevos RAL al ERBPI, para manejar nuevos robots o simuladores, sin la necesidad de recompilar los módulos CORE y GUI.

Por otra parte, esto permite a los módulos CORE y GUI cargar distintos RALs en tiempo de ejecución sin la necesidad de tener que reiniciar la aplicación.

La interfaz general que brinda el módulo RAL es la siguiente:

- *inicializarRAL:* Inicializa la conexión con el robot, real o simulado, que se utilizará.
- *finalizarRAL:* Finaliza la conexión con el robot.
- *getFrecuenciaTrabajo:* Devuelve la frecuencia de trabajo y comunicación apropiada para el robot.
- *getListaSensores:* Devuelve la lista de sensores que posee el robot.
- *getListaActuadores:* Devuelve la lista de actuadores que posee el robot.
- *getEstadoSensores:* Devuelve la lista con los valores de cada sensor que posee el robot en este momento.
- *setEstadoActuadores:* Define los valores de cada actuador que posee el robot.

En la actualidad, se encuentran implementados los módulos RAL específicos para los robots Khepera y ExaBot, y para los simuladores Yaks y ExaSim. De esta forma, el ERBPI puede controlar estas cuatro plataformas robóticas.

5.4.2. Normalización de valores

Una de las funcionalidades más importantes del RAL, además de establecer la correcta conexión con el robot, es abstraer al módulo CORE de los valores reales de los sensores y actuadores de cada robot, realizando la normalización de los mismos.

Todas las salidas de las funciones y los valores de los actuadores y sensores son normalizados al rango de valores $[-100, 100]$ para funciones y actuadores y $[0, 100]$ para sensores. Este rango representa un "porcentaje de activación" para cada elemento. Esta normalización de los datos permite abstraer los valores reales para cada sensor o actuador que pueden ser de distintos rangos y escalas en cada caso. A la vez, permite al CORE funcionar independientemente de la implementación de cada sensor o actuador particular y que un comportamiento pueda ser utilizado para varios robots con diferentes sensores o actuadores.

Cada plataforma, robot o simulador, posee implementaciones particulares y distintas entre sí de sensores y actuadores. Por ejemplo, el ExaBot posee un anillo de sensores infrarrojos de proximidad (telémetros) que por sus características

devuelven valores entre $[0, 157]$ que indican una medida de tiempo transcurrido, entre la emisión del haz y su retorno, que puede traducirse en una unidad de distancia. A la vez, la relación entre la distancia y el valor que se obtiene es no-lineal e inversamente proporcional. Es decir, un valor menor implica estar viendo un objeto "muy cerca" y un valor mayor un objeto "muy lejos". Por lo tanto, resulta necesario para el ERBPI poder expresar de manera práctica y sencilla estos valores para que puedan ser utilizados en los comportamientos y se correspondan de alguna manera con los valores expresados por los sensores de otras plataformas de robots. De esta forma, el RAL particular del ExaBot se encarga de linealizar, invertir y transformar estos valores al rango $[0, 100]$ representando los valores de $0\,\%$ y $100\,\%$ de activación respectivamente, donde $0\,\%$ implica no ver ningún objeto y $100\,\%$ estar viendo uno "muy cerca".

Otro ejemplo es el de los actuadores, donde cada plataforma de robot tiene su implementación concreta que implican valores de máxima y mínima velocidad para las ruedas. Mientras la gran mayoría de los robots establece los valores de sus actuadores como un valor de velocidad individual para cada rueda, con distintos rangos y escalas dependiendo el robot, el ExaSim establece la velocidad de cada rueda sobre la base de definirle al robot un valor de velocidad lineal y otro valor de velocidad angular. Por eso, es que también resulta necesario para el ERBPI manejar homogéneamente los valores de cada actuador para todos los robots, normalizándolos a valores en el rango $[-100, 100]$ representando valores de $-100\,\%$ de activación, máxima velocidad hacia atrás, y $100\,\%$ de activación, máxima velocidad hacia adelante.

6. Formalización del ERBPI

En esta sección realizamos la formalización del ERBPI como una máquina de estados finitos ampliada (AFSM) para que en un futuro sea posible derivar propiedades del ERBPI sobre la base de las propiedades conocidas para máquinas de estados finitos.

6.1. Comportamiento

El ERBPI posee dos interfaces gráficas para programar el comportamiento que se ejecutará en el robot, una interfaz para definir *comportamientos simples* y otra para definir el *comportamiento complejo* como composición de los *comportamientos simples* anteriores.

6.1.1. Comportamiento Simple

La interfaz para definir el comportamiento simple, como muestra la Figura 29, permite establecer un conjunto con repeticiones de cuatro funciones bien definidas a las cuales pueden conectarse todos los sensores que posee el robot y, a la vez, todas estas funciones pueden conectarse a los actuadores del mismo. De esta forma, el comportamiento simple define la función de salida de la máquina de estados finitos.

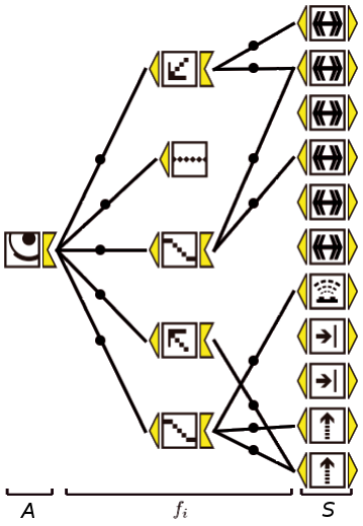

Figura 29: Interfaz del ERBPI para la programación del comportamiento simple, ejemplo para la rueda izquierda del robot. La programación del comportamiento simple define la función de salida de la máquina de estados finitos.

6.1.2. Comportamiento Complejo

La interfaz para definir el comportamiento complejo, como muestra la Figura 30, permite definir los distintos estados o nodos, el estado inicial q_0 y realizar las conexiones entre ellos definiendo cada función de transición entre estados. Cada estado puede definirse con la interfaz para comportamientos simples. De esta forma, el comportamiento complejo define las transiciones entre estados de la máquina de estados finitos.

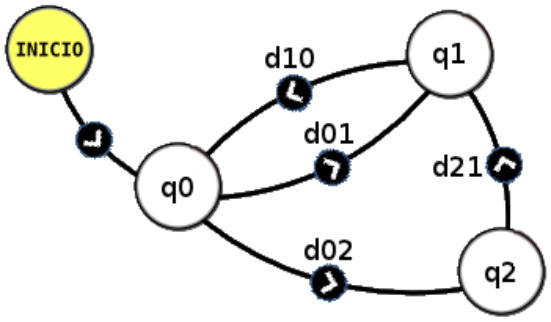

Figura 30: Interfaz del ERBPI para la programación del comportamiento complejo. La programación del comportamiento complejo define las transiciones entre estados de la máquina de estados finitos.

6.2. Formalización

Podemos formalizar la composición de los comportamientos simples, el comportamiento complejo resultante del ERBPI, como un autómata finito determinístico (DFA). En particular, definiremos una *Máquina de Mealy* [50], un caso general con *salidas* asociadas a los *estados* y a la *entrada*.

De esta forma, el comportamiento complejo resultante M queda definido por la 6-upla:

$$M =< Q, \Sigma, \Delta, \delta, \omega, q_0 >$$

donde:

- Q: conjunto de estados.
- Σ: alfabeto de entrada.
- Δ: alfabeto de salida.
- δ: función de transición.
- ω: función de salida.
- q_0: estado inicial.

6.2.1. Timers y Contadores en el ERBPI

El ERBPI brinda la posibilidad de configurar temporizadores (*timers*) y contadores. En las AFSMs los *timers* y *contadores* se encuentran asociados a los estados. Sin embargo, por razones didácticas, la interfaz del ERBPI utiliza los *timers* y *contadores* en las transiciones.

Esta decisión, relacionada al diseño del ERBPI, se fundamenta en que en la definición de los comportamientos simples (estados) no aparezcan los *timers* y *contadores*. Esto permite mantener en la programación de los estados el concepto de comportamientos reactivos según el modelo de Braitenberg, ignorando en el comportamiento simple cómo se encuentra integrado dentro de un comportamiento complejo (máquina de estados). Además, esta decisión permite reutilizar los comportamientos simples en la composición de múltiples comportamientos complejos, donde sí aparece la noción de *timers* y *contadores* globales para toda la programación del mismo. Por esto mismo, en el ERBPI la configuración de *timers* y *contadores* se realiza asociado a las transiciones, en lugar de asociado a los estados como en las AFSMs.

Sin embargo, el ERBPI brinda la posibilidad de establecer estados que no realizan nada, o sea, que no producen salidas en Δ, y transiciones que se realizan inmediatamente al no tener condiciones definidas, o sea, transiciones λ. Por lo tanto, existe una equivalencia entre el ERBPI, con *timers* y *contadores* asociados a las transiciones, y una AFSM con *timers* y *contadores* asociados a los estados.

En la Figura 31 puede verse un ejemplo sencillo de esta equivalencia. La máquina del ERBPI con dos estados, q_0 y q_1, y una transición d_{01} desde q_0 a q_1 donde se modifican los *timers* y *contadores*. La transición d_{01} se separa en dos nuevas transiciones, d_{02} y λ, donde la condición de activación de d_{02} es la condición de activación de d_{01}, pero d_{02} ya no contiene la modificación de los *timers* y *contadores*. A la vez, se agrega un nuevo estado q_2 donde se realizan las modificaciones de los *timers* y *contadores*. La secuencia entre transiciones y estados d_{02}-q_2-λ reemplaza a la transición d_{01} obteniendo una AFSM con los *timers* y *contadores* asociados a los estados.

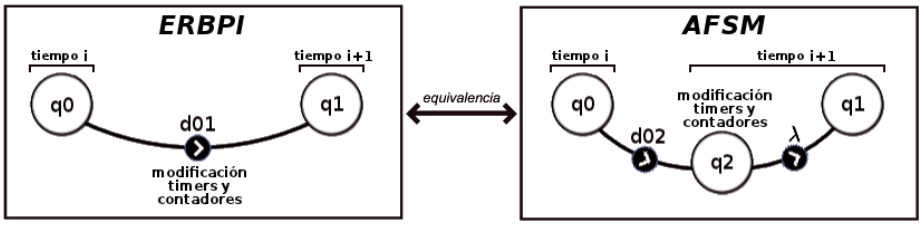

Figura 31: Equivalencia del ERBPI y AFSM para modificación de timers y contadores. La transición d_{01} se separa en d_{02} y λ. La transición d_{02} tiene la misma condición de activación que d_{01} pero no modifica los *timers* y *contadores*. El nuevo estado q_2 contiene las modificaciones de *timers* y *contadores*. La transición λ permite que en el $tiempo_{i+1}$ prosiga la ejecución de q_1 junto a las modificaciones de los *timers* y *contadores* ahora asociados a los estados.

En adelante modelamos a los *timers* y *contadores* dentro de los estados, con lo que la función de transición δ quedará definida como se espera para una AFSM, como veremos más adelante.

6.2.2. Alfabeto de entrada (Σ)

Σ está definido por la tupla de sensores del robot, timers y contadores.

$$\Sigma = S \times T \times C$$

donde S tupla de sensores, $S =< s_0, s_1, \ldots, s_{|S|-1} >$
donde T tupla de timers, $T =< timer_0, \ldots, timer_{|T|-1} >$
donde C tupla de contadores, $C =< cont_0, \ldots, cont_{|C|-1} >$
con $s_i \in [0; 100]$, $timer_i, cont_i \in [0; maxint]$
y $maxint$ el máximo número entero representable.

6.2.3. Alfabeto de salida (Δ)

Δ está definido por la tupla de actuadores del robot, timers y contadores. En nuestro caso los actuadores son dos, izquierda (L) y derecha (R).

$$\Delta = A \times T \times C$$

donde A tupla de actuadores, $A =< L, R >$
donde T y C definidos anteriormente.
con $L, R \in [-100; 100]$

Los alfabetos de entrada y salida quedan definidos como tuplas de valores en los rangos antes mencionados. Es importante tener en cuenta que los alfabetos Σ y Δ no contienen relación entre sí. La formalización que describimos permite cualquier combinación resultante de todos los valores definidos en los rangos. En cambio, el ERBPI produce un subconjunto de estas combinaciones en función del comportamiento definido. No establecemos estas restricciones en la formalización de la máquina de estados. Por ejemplo, un alfabeto de entrada y salida posible sería:

$$< \underbrace{50, 10, \ldots, 99}_{S}, \underbrace{1, \ldots, 5}_{T}, \underbrace{0, \ldots, 3}_{C} > \quad \in \quad \Sigma$$

$$< \underbrace{30, -20}_{A}, \underbrace{3, \ldots, 8}_{T}, \underbrace{6, \ldots, 7}_{C} > \quad \in \quad \Delta$$

6.2.4. Función de transición (δ)

δ determina, dado el estado actual y $\sigma \in \Sigma$, cuál es el nuevo estado actual.

$$\delta : Q \times \Sigma \longrightarrow Q$$

El ERBPI permite utilizar para la programación de las funciones de transición cinco operadores lógicos bien definidos: menor estricto (e_1), menor igual (e_2), igual (e_3), mayor igual (e_4) y mayor estricto (e_5). Además, para terminar de definir las condiciones de las transiciones, es necesario una operador más que no se encuentra explícito pero que el ERBPI utiliza implícitamente para definir los elementos sobre los que no se establecen condiciones: (e_6). Entonces:

- $e_1(x,k) = \begin{cases} 1 & \text{si } x < k \\ 0 & \text{en otro caso} \end{cases}$
- $e_2(x,k) = \begin{cases} 1 & \text{si } x \leq k \\ 0 & \text{en otro caso} \end{cases}$
- $e_3(x,k) = \begin{cases} 1 & \text{si } x = k \\ 0 & \text{en otro caso} \end{cases}$
- $e_4(x,k) = \begin{cases} 1 & \text{si } x \geq k \\ 0 & \text{en otro caso} \end{cases}$
- $e_5(x,k) = \begin{cases} 1 & \text{si } x > k \\ 0 & \text{en otro caso} \end{cases}$
- $e_6(x) = 1$

con $x \in \bigcup_{i=0}^{N-1} \{\pi_i(\sigma)\}$, $\sigma \in \Sigma$

y $\pi_i(\sigma)$ i-ésimo elemento de σ y $N = |S| + |T| + |C|$ longitud de σ.

Ahora, definimos a δ como:

$$\delta(q_i, \sigma) = q_j \iff d_{ij}(\sigma) = 1$$

con $q_i, q_j \in Q$, $\sigma \in \Sigma$ y d_{ij} definido como:

$$d_{ij}(\sigma) = e_{l_0}(\pi_0(\sigma)) \wedge e_{l_1}(\pi_1(\sigma)) \wedge \cdots \wedge e_{l_{N-1}}(\pi_{N-1}(\sigma))$$

con $i, j \in [0; |Q| - 1]$ y $l_h \in \{1, 2, 3, 4, 5, 6\}$

Por último, el ERBPI permite establecer más de una transición saliente desde un nodo o estado. Por ejemplo, con tres estados q_1, q_2 y q_3 se pueden establecer las transiciones d_{12} y d_{13}. Estas transiciones pueden cumplirse simultáneamente, pero el ERBPI selecciona determinísticamente una de ellas. De esta forma, para todo estado $q_i \in Q$ y $\sigma \in \Sigma$ existe un único estado $q_j \in Q$ tal que $d_{ij}(\sigma) = 1$.

6.2.5. Función de Salida (ω)

ω determina el elemento de Δ en función del estado actual y $\sigma \in \Sigma$.

$$\omega : Q \times \Sigma \longrightarrow \Delta$$

Para poder definir completamente ω, definimos primero un conjunto de funciones f, t y c para los actuadores, timers y contadores respectivamente.

Función de Actuadores (f_{L_i} y f_{R_i})

El ERBPI permite utilizar para la programación del comportamiento simple cuatro funciones bien definidas: excitatoria (g_1), inhibitoria (g_2), partida (g_3) y constante (g_4). Además, para terminar de definir la programación del comportamiento simple, son necesarias dos funciones más que no se encuentran explícitas pero que el ERBPI utiliza implícitamente: normalización (g_0) y nula (g_5). Sea:

- $g_0(x) = \begin{cases} 100 & \text{si } x > 100 \\ -100 & \text{si } x < -100 \\ x & \text{en otro caso} \end{cases}$
- $g_1(x) = x$
- $g_2(x) = -x$
- $g_3(x, a, b) = \begin{cases} a_y & \text{si } x \leq a_x \\ b_y & \text{si } x \geq b_x \\ \dfrac{a_y - b_y}{a_x - b_x}(x - a_x) + a_y & \text{si } a_x < x < b_x \end{cases}$
  con a y b puntos de la forma $a = (a_x, a_y)$ y $b = (b_x, b_y)$
- $g_4(x, cte) = cte$
- $g_5(x) = \begin{cases} x & \text{si está conectado} \\ 0 & \text{en otro caso} \end{cases}$

con $x \in \bigcup_{i=0}^{|S|-1} \{\pi_i(\sigma)\}$ y $\sigma \in \Sigma$

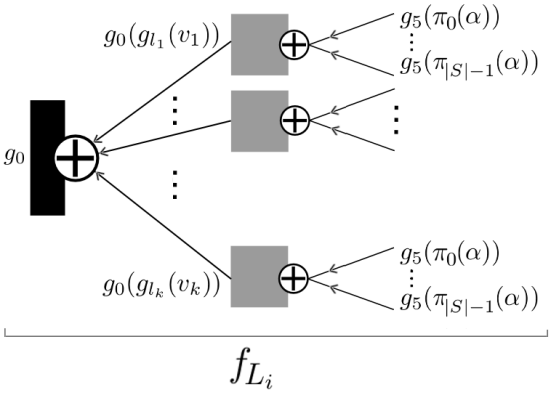

Figura 32: Esquema de f_{L_i}, f_i para L, determinado por la programación del comportamiento simple en el ERBPI. Puede verse la correlación de este esquema con el de la Figura 29 en la interfaz del ERBPI.

Además, el ERBPI permite definir en cada estado funciones distintas para cada actuador. Entonces, para facilitar la notación, definimos a la función de actuadores para un estado $f(q_i, \alpha)$ como $f_i(\alpha)$.

Donde $\alpha \in \pi_{0,\ldots,|S|-1}(\Sigma)$, la proyección de los sensores de Σ.

Así, separamos a $f_i(\alpha)$ como $f_{L_i}(\alpha)$ y $f_{R_i}(\alpha)$, la funciones f asociadas al estado $q_i \in Q$, para el actuador izquierdo L y derecho R, de la siguiente manera:

$$f_i : \pi_{0,\ldots,|S|-1}(\Sigma) \quad \longrightarrow \quad [-100;100] \times [-100;100]$$

$$f_{L_i} : \quad < s_0, s_1, \ldots, s_{|S|-1} > \quad \longrightarrow \quad l \quad , \quad l \in [-100;100]$$

$$f_{R_i} : \quad < s_0, s_1, \ldots, s_{|S|-1} > \quad \longrightarrow \quad r \quad , \quad r \in [-100;100]$$

$f_{L_i} = g_0(\quad g_0(g_{l_1}(v_1)) \quad + \quad g_0(g_{l_2}(v_2)) \quad + \quad \ldots \quad + \quad g_0(g_{l_k}(v_k)) \quad)$
con $l_j \in \{1,2,3,4\}$

$f_{R_i} = g_0(\quad g_0(g_{m_1}(w_1)) \quad + \quad g_0(g_{m_2}(w_2)) \quad + \quad \ldots \quad + \quad g_0(g_{m_k}(w_k)) \quad)$
con $m_j \in \{1,2,3,4\}$

donde v_j y w_j son de la forma: $\sum_{i=0}^{|S|-1} g_5(\pi_i(\alpha))$

De esta forma, la función g_5 nos permite especificar cuáles, y cuáles no, de los $|S|$ sensores del robot incidirán en la entrada de la funciones g_{l_j} y g_{m_j} especificadas.

Cada una de las g_{l_j} y g_{m_j} son normalizadas con g_0, son sumadas entre sí y el resultado es normalizado nuevamente con g_0. Este último resultado es f_{L_i} y f_{R_i}.

En la Figura 32 puede verse un esquema de cómo el ERBPI define f_i.

Función de Timers y Contadores (t_i y c_i)

El ERBPI permite utilizar para la modificación de timers una única función: resetear (η_1). Para la modificación de contadores tres funciones bien definidas: resetear (η_2), incrementar (η_3) y decrementar (η_4). Además, para terminar de definir la modificación de los timers y contadores, es necesaria una función más que no se encuentra explícita pero que el ERBPI utiliza implícitamente: identidad (η_5). Sea:

- $\eta_1(x) = 0$
- $\eta_2(y) = 0$
- $\eta_3(y) = y + 1$
- $\eta_4(y) = y - 1$
- $\eta_5(z) = z$

$$\text{con } x \in \bigcup_{i=|S|}^{|S|+|T|-1} \{\pi_i(\sigma)\} \quad , \quad y \in \bigcup_{i=|S|+|T|}^{|S|+|T|+|C|-1} \{\pi_i(\sigma)\}$$

$$\text{y } z \in \bigcup_{i=|S|}^{|S|+|T|+|C|-1} \{\pi_i(\sigma)\}$$

Entonces, de igual forma que con la función de actuadores, definimos a $t(q_i, \mu)$ y $c(q_i, \nu)$ como las funciones $t_i(\mu)$ y $c_i(\nu)$.

Donde $\mu \in \pi_{|S|,\ldots,|S|+|T|-1}(\Sigma)$, la proyección de los timers de Σ, y $\nu \in \pi_{|S|+|T|,\ldots,|S|+|T|+|C|-1}(\Sigma)$, la proyección de los contadores de Σ.

Así, definimos las funciones de timers t_i y contadores c_i asociadas al estado $q_i \in Q$ de la siguiente manera:

$$t_i : \quad \pi_{|S|,\ldots,|S|+|T|-1}(\Sigma) \quad \longrightarrow \quad \pi_{|S|,\ldots,|S|+|T|-1}(\Sigma)$$

$$t_i : \quad < timer_0, \ldots, timer_{|T|-1} > \quad \longrightarrow \quad < timer_0, \ldots, timer_{|T|-1} >$$

$$c_i : \quad \pi_{|S|+|T|,\ldots,|S|+|T|+|C|-1}(\Sigma) \quad \longrightarrow \quad \pi_{|S|+|T|,\ldots,|S|+|T|+|C|-1}(\Sigma)$$

$$c_i : \quad < cont_0, \ldots, cont_{|C|-1} > \quad \longrightarrow \quad < cont_0, \ldots, cont_{|C|-1} >$$

Entonces:

$$t_i(\mu) = \eta_{l_0}(\pi_0(\mu)) \times \eta_{l_1}(\pi_1(\mu)) \times \cdots \times \eta_{l_{|T|-1}}(\pi_{|T|-1}(\mu))$$
$$\text{con } l_j \in \{1, 5\}$$

$$c_i(\nu) = \eta_{m_0}(\pi_0(\nu)) \times \eta_{m_1}(\pi_1(\nu)) \times \cdots \times \eta_{m_{|C|-1}}(\pi_{|C|-1}(\nu))$$
$$\text{con } m_j \in \{2, 3, 4, 5\}$$

De esta forma, la función η_5 nos permite especificar cuáles timers y contadores no son modificados. Las funciones η_1 cuáles timers son reseteados. Las funciones η_2, η_3 y η_4 cuáles contadores son modificados.

Función de Salida Resultante (ω)

Ahora, podemos definir la función de salida ω en función de f_i, t_i y c_i, de la siguiente forma:

$$\omega : Q \times \Sigma \longrightarrow \Delta$$

$$\omega(q_i, \sigma) \quad = \quad < \underbrace{l}_{f_{L_i}(\alpha)} \, , \, \underbrace{r}_{f_{R_i}(\alpha)} \, , \, \underbrace{timer_0, \ldots, timer_{|T|-1}}_{t_i(\mu)} \, , \, \underbrace{cont_0, \ldots, cont_{|C|-1}}_{c_i(\nu)} >$$

7. Resultados

El LRSE viene desarrollando diversas actividades y talleres de Robótica Educativa orientados principalmente a alumnos de los últimos años de la escuela media.

En estas actividades se utilizan los conceptos básicos de la robótica basada en comportamientos partiendo del modelo de Braitenberg. Sobre la base del planteo de resolución de problemas, se estimula a los participantes a programar comportamientos estableciendo conexiones entre los sensores y los actuadores del robot. También se motiva a los alumnos a vincular los comportamientos alcanzados con características humanas y/o animales y a ejercitar el método científico, elaborando hipótesis para resolver el problema dado, experimentando y contrastando con la realidad y finalmente sacando conclusiones para mejorar los comportamientos logrados en los robots.

7.1. Experimentos con comportamientos simples

Los problemas más comúnmente planteados para resolver mediante el modelo de Braitenberg abarcan desde la evasión de obstáculos, el seguimiento de una pared, de una fuente de luz o de una línea, hasta la resolución de un laberinto.

A continuación, describimos la resolución a algunos de estos problemas planteados utilizando la nueva herramienta tanto en simuladores como en los robots reales.

7.1.1. Avanzar evitando obstáculos

Este problema presupone que, además de evitar los obstáculos, el robot debe avanzar constantemente en algún sentido. Entonces, es necesario establecer las conexiones entre sensores y actuadores para que el robot avance y, al mismo tiempo, las conexiones para que evite los obstáculos que se le presenten.

Una vez interiorizados en el modelo de los Vehículos de Braitenberg, resulta sencillo resolver este problema. Para que el robot avance, se realizan conexiones de funciones constantes a cada actuador. Para que el robot evite obstáculos utilizando los sensores de distancia, se pueden establecer las conexiones con funciones excitatorias o inhibitorias.

Como puede verse en los ejemplos que detallamos a continuación en los simuladores y robots, es suficiente para resolver este problema la combinación de los vehículos simples de Braitenberg que utilizan las conexiones excitatorias sin cruzar e inhibitorias cruzadas simultáneamente.

Simulador Yaks

En la Figura 33 puede verse este ejemplo en el simulador Yaks. De los ocho sensores de distancia que posee el robot, se utilizan los seis frontales. Se observan dos funciones constantes conectadas a cada rueda del robot. En la Figura 33 han

sido coloreados en rojo y azul los sensores utilizados, tanto en el esquema del robot como en el escritorio de trabajo. Los actuadores y funciones constantes han sido coloreados en verde.

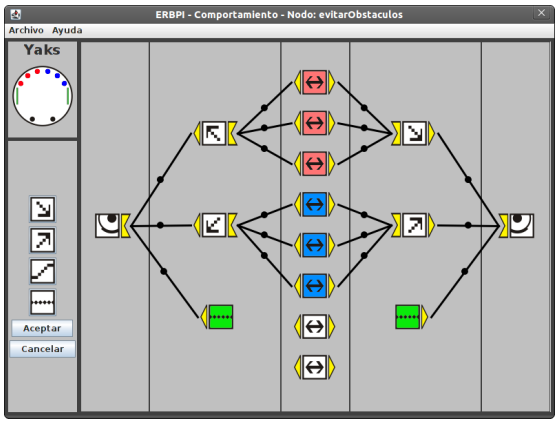

Figura 33: Esquema de comportamiento simple para avanzar evitando obstáculos con el simulador Yaks.

Para clarificar cómo funciona esta solución, supongamos que el robot encuentra un obstáculo posicionado a su izquierda. Entonces, los tres sensores de la izquierda del robot se activan. Como estos sensores están conectados con una función excitatoria al actuador izquierdo y con una función inhibitoria al actuador derecho, entregan más energía al actuador izquierdo y menos al derecho, lo que hará girar el robot hacia la derecha evitando el obstáculo. De forma análoga, se encuentran definidas las conexiones para los tres sensores ubicados a la derecha del robot. Las dos funciones constantes a cada actuador resultan en que el robot avance constantemente.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento simple para avanzar evitando obstáculos con el simulador Yaks.

Simulador ExaSim

En la Figura 34 puede verse el mismo ejemplo de comportamiento simple para avanzar evitando obstáculos con el simulador ExaSim. A diferencia del Yaks, el ExaSim posee en el frente sólo tres sensores, por lo que los dos telémetros para medir distancias ubicados en las esquinas en la parte frontal del robot son conectados análogamente al esquema anterior para el Yaks. El telémetro frontal es conectado de forma excitatoria al actuador derecho para priorizar el giro a la izquierda en el caso que se encuentre con un objeto de forma frontal. Los distintos componentes han sido coloreados de la misma forma que en el ejemplo anterior.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento simple para avanzar evitando obstáculos con el simulador ExaSim.

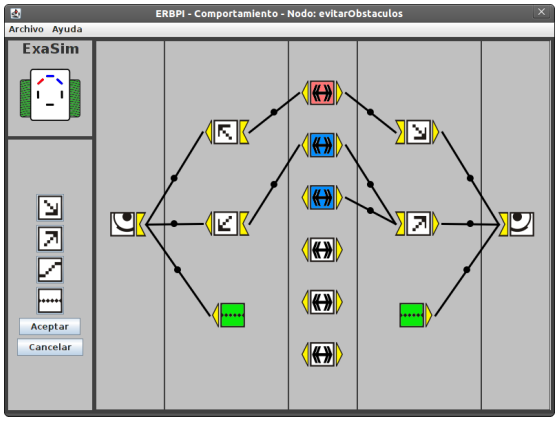

Figura 34: Esquema de comportamiento simple para avanzar evitando obstáculos con el simulador ExaSim.

Robot ExaBot

El mismo esquema de conexiones realizado para el ExaSim puede establecerse para el ExaBot, de forma que sólo cambia la plataforma sobre la que ejecutaremos el comportamiento. El esquema de conexiones para el ExaBot se corresponde con el de la Figura 34.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento simple para avanzar evitando obstáculos con el robot ExaBot.

7.1.2. Seguir la línea en el piso

Este es uno de los problemas más comunes planteados en las actividades de Robótica Educativa, denominado comúnmente como *robot seguidor de línea*. Existen incluso diversas competencias de robótica con categorías de *Carreras* que consisten en robots que compiten entre sí corriendo una carrera sobre una pista marcada con una línea. En la Argentina, se viene desarrollando desde hace varios años la *Competencia Nacional de Robótica* en Bahía Blanca, Buenos Aires, que incluye esta categoría [51]. El LRSE ha participado en varias oportunidades en esta competencia, en la categoría de *carreras* y *mini-sumo*.

Para resolver este problema el robot debe mantenerse sobre la línea y al mismo tiempo avanza constantemente.

A diferencia del comportamiento anterior, esta vez utilizamos conexiones entre sensores y actuadores con funciones excitatorias cruzadas. Esto implica que, si el robot detecta la línea con el sensor izquierdo, quiere decir que la línea está quedando ubicada a la izquierda del cuerpo del robot, por lo que es necesario entregarle más energía al actuador derecho para que el robot gire hacia la izquierda y vuelva a ubicar la línea debajo del cuerpo del robot. De forma análoga sucede con el sensor de detección de línea de la derecha.

Por el momento, el ExaBot es el único robot que maneja el ERBPI que posee sensores de detección de línea, de modo que ejemplificaremos este comportamiento únicamente con dicho robot.

Robot ExaBot

En la Figura 35 puede verse un ejemplo de comportamiento simple para seguir la línea en el piso con el robot ExaBot. Se conectan dos funciones constantes a cada rueda del robot para que avance y los dos sensores de detección de línea se conectan con funciones partidas cruzadas a cada actuador para mantenerse sobre la línea.

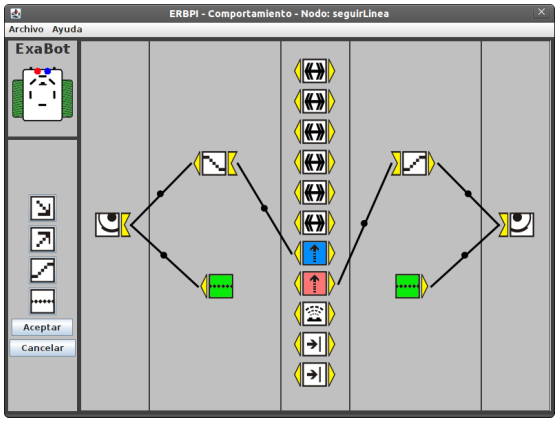

Figura 35: Esquema de comportamiento simple para seguir la línea en el piso con el robot ExaBot.

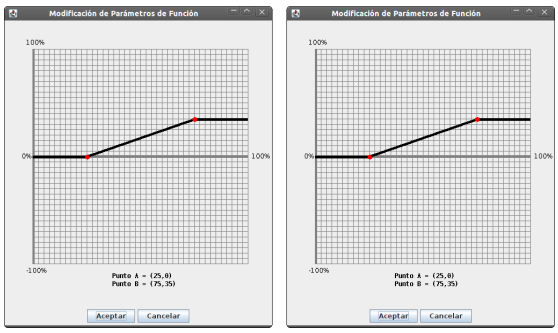

Figura 36: Configuración de funciones partidas utilizadas en el esquema para el sensor izquierdo y derecho en la Figura 35 para seguir la línea en el piso con el robot ExaBot.

Para programar este comportamiento utilizamos funciones partidas y no simplemente excitatorias por la siguiente razón. Los sensores de detección de línea indican el valor 0 % cuando no están viendo una línea blanca y 100 % en caso de verla. No poseen valores intermedios en el rango $[0\,\%, 100\,\%]$ por lo que

resulta indistinto el tiempo que el sensor lleva detectando la línea. Por lo tanto, no se puede hallar el grado de desviación respecto de la línea, con lo que la utilización de funciones excitatorias o inhibitorias resulta en movimientos de corrección demasiado "bruscos".

La velocidad con la que el robot avanza sobre la línea se vuelve también un parámetro importante a tener en cuenta. Por esto mismo, las funciones constantes deben configurarse para garantizar una velocidad que permita al robot seguir la línea correctamente cuando existen curvas muy pronunciadas. En la Figura 36 puede verse el esquema de las dos funciones partidas utilizadas en el esquema de conexiones.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento simple para seguir la línea en el piso con el robot ExaBot.

7.1.3. Seguir la pared y entrar en la primera puerta

Este es otro de los problemas más utilizados en las actividades de Robótica Educativa. Se lo conoce comúnmente como el "comportamiento del borracho". Este problema intenta reproducir el comportamiento de una persona en estado de ebriedad que avanza por un pasillo de habitaciones de un hotel, guiándose con la pared, e intentando encontrar la puerta de su habitación. El comportamiento consiste en avanzar manteniendo una distancia cercana a la pared y entrar en la primera puerta que encuentre. El ejemplo se realiza con la pared de la izquierda, pero podría hacerse con la pared de la derecha o incluso con ambas paredes simultáneamente.

En principio resulta en un problema sencillo que inmediatamente se vuelve difícil ya que se requiere avanzar manteniendo una distancia prudencial a la pared, ni demasiado lejos ni demasiado cerca y sin incurrir en oscilaciones desestabilizantes, utilizando únicamente el sensor de distancia posicionado a la izquierda del robot. Por esto mismo, resulta difícil de resolver únicamente con funciones excitatorias o inhibitorias, obligando a utilizar las funciones partidas para su resolución.

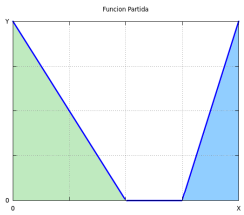

Figura 37: Ejemplo de una función partida para seguir una pared a una determinada distancia. El segmento inferior marcado en color verde indica el rango de valores menores del sensor que se traducen en objetos a distancias mayores. El segmento superior marcado en azul indica el rango de valores mayores del sensor que se traducen en objetos a distancias menores.

Bajo esta premisa, mantener el robot constantemente a una distancia pre-

determinada de la pared mientras éste avanza, se traduce en que sus actuadores deberán recibir energía constante mientras el sensor de la izquierda mantenga sus valores de sensado en un rango determinado en el centro. Por otro lado, cuando este sensor entregue valores fuera de este rango, en su extremo inferior o superior, deberá entregar mayor energía a un actuador que a otro dependiendo si se encuentra muy lejos de la pared (extremo inferior de los valores) o muy cerca (extremo superior de los valores). En una primera aproximación, se puede pensar en una función partida de forma de representar el comportamiento antes mencionado. En la Figura 37 se muestra un ejemplo de esta función.

Para poder utilizar las funciones partidas provistas por el ERBPI, basta con descomponer la función de la Figura 37 en tres casos:

- *Segmento Intermedio:* en este caso, el robot se encuentra a la distancia requerida de la pared, por lo que el valor de esta función para los actuadores es 0 % y el robot se mueve según los valores de las funciones constantes para garantizar su avance.
- *Segmento Inferior:* en este caso, el robot se encuentra muy lejos de la pared ubicada a la izquierda ya que los valores del sensor son los menores de su rango, por lo que esta función entrega más energía al actuador derecho que al izquierdo para que el robot gire a izquierda y se acerque a la pared.
- *Segmento Superior:* en este caso, el robot se encuentra muy cerca de la pared ubicada a la izquierda ya que los valores del sensor son los mayores de su rango, por lo que esta función entrega más energía al actuador izquierdo que al derecho para que el robot gire a derecha y se aleje de la pared.

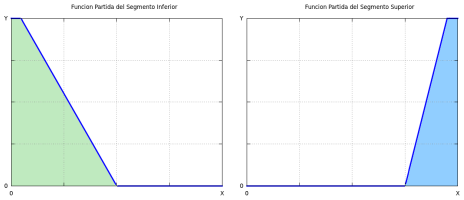

Figura 38: Ejemplo de descomposición de la función de la Figura 37 en dos funciones partidas según el esquema de Vehículos de Braitenberg.

De esta forma, se puede realizar un esquema, según los vehículos de Braitenberg, donde cada uno de los tres casos se corresponda con una función y su respectiva conexión al sensor. En la Figura 38 pueden verse los esquemas de estas dos funciones partidas correspondientes a los segmentos *inferior* y *superior* detallados anteriormente. Puede verse además, que estas dos funciones partidas no son otra cosa que el desdoblamiento de la función pensada originalmente. El segmento *intermedio* se determina con funciones constantes.

Simulador ExaSim

En la Figura 39 puede verse un ejemplo de comportamiento simple para seguir la pared y entrar en la primera puerta con el simulador ExaSim. Se conectan una función constante a cada rueda del robot y una función partida desde el sensor izquierdo a cada rueda. Estas conexiones se traducen en que el robot, además de avanzar constantemente, intentará mantener una distancia prudencial a la pared determinada por las dos funciones partidas. Al encontrar el primer lugar donde no haya pared, una puerta, entrará.

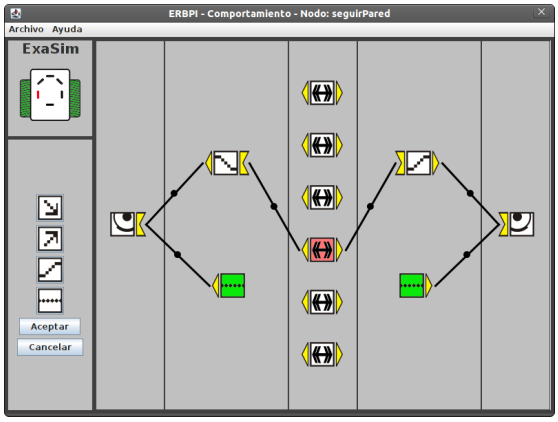

Figura 39: Esquema de comportamiento simple para seguir la pared y entrar en la primera puerta con el simulador ExaSim.

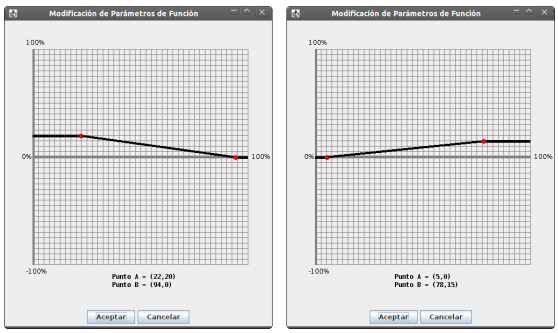

Figura 40: Configuración de funciones partidas utilizadas en el esquema para el actuador derecho e izquierdo en la Figura 39 para seguir la pared y entrar en la primera puerta con el simulador ExaSim.

Como puede verse en la Figura 39 las conexiones respetan el esquema explicado anteriormente. A la vez, las funciones partidas, que pueden verse en la Figura 40, han sido configuradas de manera de ajustarlas a las particularidades de los sensores del ExaSim para mantener una distancia razonable a la pared y lograr que el robot se dirija a través de la puerta ni bien ésta aparezca.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento simple con el simulador ExaSim.

Simulador Yaks

En la Figura 41 puede verse la resolución de este mismo comportamiento con el simulador Yaks. El esquema es análogo al del ejemplo anterior, pero esta vez se utiliza el sensor infrarrojo de proximidad para medir distancia ubicado más a la izquierda en el robot.

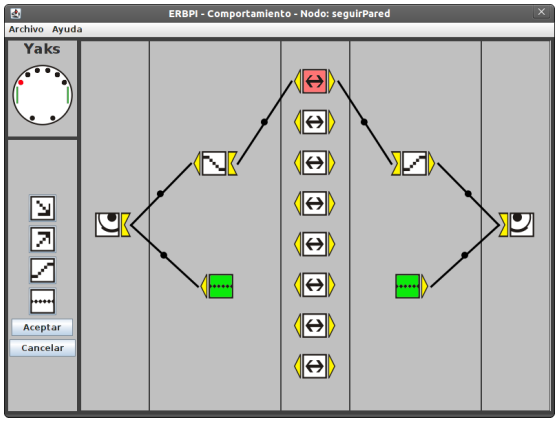

Figura 41: Esquema de comportamiento simple para seguir la pared y entrar en la primera puerta con el simulador Yaks.

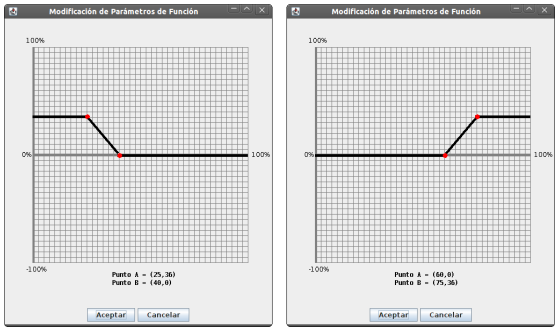

Figura 42: Configuración de funciones partidas utilizadas en el esquema para el actuador derecho e izquierdo en la Figura 41 para seguir la pared y entrar en la primera puerta con el simulador Yaks.

En la Figura 42 puede verse el esquema de las funciones partidas configuradas de manera que se ajusten a los sensores del Yaks para mantener una distancia razonable a la pared y lograr que el robot se dirija a través de la puerta ni bien ésta aparezca.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento simple para seguir la pared y entrar en la primera puerta con el simulador Yaks.

Robot ExaBot

El mismo esquema de conexiones realizado para el ExaSim puede establecerse para el ExaBot, de forma que sólo cambia la plataforma sobre la que ejecutamos el comportamiento. El esquema de conexiones para el ExaBot se corresponde con los de las Figuras 39 y 40.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento simple para seguir la pared y entrar en la primera puerta con el robot ExaBot.

7.2. Experimentos con comportamientos complejos

Hasta este momento, hemos descripto y resuelto algunos problemas simples utilizando la nueva herramienta y siguiendo únicamente el modelo de los Vehículos de Braitenberg. En adelante plantearemos más condiciones a estos problemas para que resulten en problemas más complejos.

¿Qué sucedería, por ejemplo, si en el comportamiento de seguir una línea en el piso se agregara la condición de que se desconoce la ubicación inicial del robot y, por lo tanto, no se sabe dónde se encuentra ubicada la línea inicialmente? Si el robot no comienza encarrilado sobre la línea, ¿cómo se diferenciaría el caso de que la línea esté ubicada justo entre los dos sensores, lo que hace que ambos indiquen que no ven la línea, con comenzar completamente fuera de la línea donde los sensores indican lo mismo?

Como otro ejemplo de estas nuevas condiciones que resultan en comportamientos más complejos, se puede plantear, para el comportamiento de seguir la pared y entrar en la primera puerta, que podrían existir obstáculos en medio del pasillo que serían necesarios esquivar para llegar a la puerta.

Un ejercicio que los alumnos suelen realizar en las experiencias de Robótica Educativa es intentar resolver estos nuevos problemas únicamente siguiendo el modelo de los Vehículos de Braitenberg. Si bien algunos de estos comportamientos son capaces de resolverse de esta forma, resulta muy complicado y "engorroso" resolverlos únicamente con un solo esquema de vehículo de Braitenberg. Otros comportamientos más complejos son directamente imposibles de resolver utilizando sólo el modelo de Braitenberg.

Por esto mismo, surge la arquitectura de subsunción como una solución sencilla y rápida para resolver estos comportamientos más complejos descomponiéndolos en varios comportamientos simples que se coordinan para resolver el problema planteado.

En adelante, frente a los nuevos problemas y comportamientos complejos planteados, intentaremos identificar subobjetivos dentro del objetivo global a cumplir para resolver el problema. De esta forma, se simplifica su resolución

coordinando comportamientos simples mediante la arquitectura de subsunción. A continuación, desarrollamos algunos ejemplos de estos problemas.

7.2.1. Seguir la línea en el piso desconociendo la ubicación inicial de la misma

Como describimos anteriormente, es posible identificar, en este comportamiento complejo, a dos comportamientos simples: *buscar la línea* y *seguir la línea*.

En el caso del comportamiento simple *seguir la línea* coincide exactamente con el problema planteado anteriormente en el Punto 7.1.2. Aquí se refleja una de las grandes ventajas que proporciona el ERBPI mediante la utilización de la arquitectura de subsunción: la reutilización e inclusión, de forma rápida y sencilla, de comportamientos simples programados con anterioridad.

De esta forma, es posible afirmar que ya se encuentra resuelto uno de los dos comportamientos simples planteados para resolver el comportamiento complejo. Sólo queda por resolver el otro, *buscar la línea*.

Para resolver el comportamiento simple *buscar la línea* puede pensarse en que el robot simplemente avance en algún sentido hasta que alguno de sus sensores detecte la línea. Una vez detectada la línea se encarrila al robot sobre la línea de forma de pasar al otro comportamiento simple de seguir la línea. De aquí se desprende también, que podemos descomponer el comportamiento *buscar la línea* en dos nuevos comportamientos simples: *buscar la línea* solamente y *encarrilar sobre la línea* al robot.

Robot ExaBot

En la Figura 43 puede verse un ejemplo del comportamiento complejo, compuesto de tres comportamientos simples, para seguir línea en el piso desconociendo la ubicación inicial de la misma, con el robot ExaBot. En este ejemplo, se utiliza la arquitectura de subsunción coordinando tres comportamientos simples: *buscar*, *encarrilar* y *seguir*.

Para simplificar el problema en un principio, se asume que el robot se encuentra inicialmente perpendicular a la línea.

Así, en el caso de *buscar*, se establecen conexiones con funciones constantes a los dos actuadores para que el robot avance en forma recta hasta que ambos sensores detecten la línea simultáneamente. Para definir las condiciones que resultan en que luego de detectar la línea el robot comience a ejecutar el comportamiento simple *encarrilar*, se define una transición con nodo origen-*buscar* y destino-*encarrilar*. La condición que se debe cumplir para que se ejecute la transición es que ambos sensores, *linea.0* y *linea.1*, sean distintos de cero como muestra la Figura 44.

El comportamiento simple *encarrilar* consiste en girar el robot en el lugar el tiempo suficiente para que éste quede encarrilado sobre la línea. Este giro queda determinado mediante dos funciones constantes de signo opuesto conectadas a

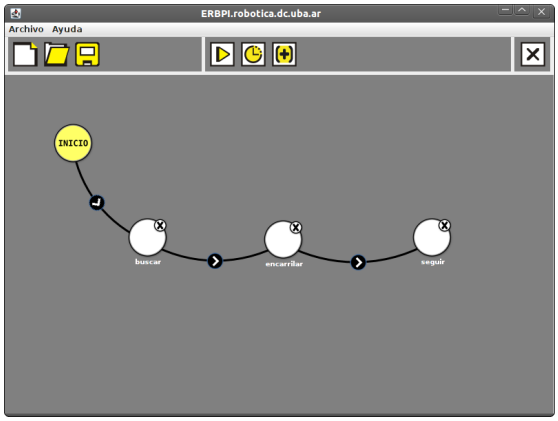

Figura 43: Esquema de comportamiento complejo para buscar y seguir la línea en el piso con el robot ExaBot, compuesto de tres comportamientos simples *buscar*, *encarrilar* y *seguir*.

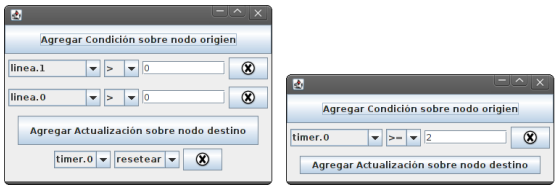

Figura 44: Ejemplo de configuración de condiciones para las transiciones del comportamiento complejo buscar y seguir la línea en el piso con el robot ExaBot de la Figura 43. A la izquierda, las condiciones y actualizaciones de la transición con nodo origen-*buscar* y destino-*encarrilar*. A la derecha, las condiciones de la transición con nodo origen-*encarrilar* y destino-*seguir*.

cada rueda del robot. Se determinó experimentalmente que con dos segundos de tiempo era suficiente para que el robot quedara encarrilado sobre la línea. Para contar este tiempo se define un temporizador *timer.0* que se resetea en la transición del comportamiento *buscar* a *encarrilar*. Esta actualización puede verse en la Figura 44. Luego, se define una transición con nodo origen-*encarrilar* y destino-*seguir* cuya condición será que el temporizador *timer.0* sea mayor o igual a dos. En la Figura 44 pueden verse las condiciones de esta transición.

El comportamiento simple *seguir* en el esquema de la Figura 43 es el mismo que el del Punto 7.1.2 y puede importarse en la aplicación con el menú para guardar y abrir comportamientos simples.

Por último, puede analizarse el caso más general, removiendo la simplificación hecha inicialmente al problema de que el robot se encuentra originalmente perpendicular a la línea. Para resolver este caso, el comportamiento *buscar* podría contemplar distintas formas de buscar la línea como girar en el lugar o avanzar girando, y no sólo avanzar en forma recta. Incluso, podrían combinarse varias de estas formas de buscar la línea estableciendo tiempos, con temporizadores, donde cada una se ejecuta durante un tiempo, de forma de ir variando la estrategia para buscar la línea y aumentar la probabilidad de encontrarla. El comportamiento *encarrilar* también podría contemplar estos distintos casos: encontrar la línea con los dos sensores simultáneamente, o sólo con un sensor. En este último caso, se podría girar el robot en el sentido correcto de forma de hacer que el sensor que no estaba viendo la línea comience a verla.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento complejo para buscar y seguir la línea en el piso con el robot ExaBot.

7.2.2. Seguir la línea en el piso evitando los obstáculos que puedan presentarse

De forma análoga al problema anterior, puede agregarse al problema de seguir una línea en el piso, que en el camino a recorrer aparezcan obstáculos a evitar. Es posible identificar en este comportamiento complejo a dos comportamientos simples: *seguir la línea* y *evitar obstáculo*.

Este problema se puede reducir a encontrar una adecuada coordinación entre los comportamientos simples ya resueltos en el Punto 7.1.2 (*seguir la línea*) y el Punto 7.1.1 (*evitar obstáculo*).

Otra posibilidad, incluso más compleja, es agregar a este problema la condición de que tampoco se conoce la posición inicial de la línea. Nuevamente, se puede resolver este problema coordinando adecuadamente los comportamientos simples *buscar*, *encarrilar* y *seguir* del problema anterior más el comportamiento simple del Punto 7.1.1 (*evitar obstáculo*).

7.2.3. Seguir la pared y entrar en la primera puerta, evitando los obstáculos que puedan presentarse

En este nuevo problema, es posible identificar dos comportamientos simples: *seguir la pared* y *evitar el obstáculo*. En el caso del comportamiento simple *seguir la pared* coincide exactamente con el problema planteado anteriormente y resuelto en el Punto 7.1.3. El comportamiento simple *evitar el obstáculo* se encuentra resuelto en el Punto 7.1.1. Sin embargo, podemos pensar en un nuevo comportamiento más simple aún, aprovechando determinadas condiciones que conocemos del problema, como por ejemplo que el robot debe seguir una pared a izquierda, con lo que los objetos que podamos encontrar deberán siempre ser esquivados por la derecha. Como el robot debe seguir nuevamente la pared luego de evitar el obstáculo, la acción de evitar el obstáculo podrá reducirse a rodear el objeto en forma circular por la derecha, de forma de esquivarlo y volver a la pared.

Así, en este caso podemos descomponer el objetivo de evitar un obstáculo en tres comportamientos simples: *girar-derecha*, *avanzar* y *girar-izquierda*. Al detectar un obstáculo con los sensores frontales mientras se está ejecutando el comportamiento *seguir la pared*, comienza a ejecutarse el comportamiento *girar-derecha*. Este comportamiento sigue hasta que el obstáculo no es más detectado por los sensores, lo que indica que el robot puede avanzar sin ser obstruido por el objeto. Luego, se ejecuta el comportamiento *avanzar* un tiempo determinado y, por último, *girar-izquierda* también durante un tiempo. Finalmente, se vuelve a la ejecución de *seguir la pared*.

En el caso que el obstáculo no haya sido completamente rodeado por el robot, al volver la ejecución al comportamiento simple *seguir la pared*, el robot vuelve a detectar un obstáculo, con lo que pasa a ejecutar los tres comportamientos simples de *girar-derecha*, *avanzar* y *girar-izquierda* para evitarlo. Así, el comportamiento complejo para evitar un obstáculo es una coordinación iterativa de estos cuatro comportamientos simples la cantidad de veces que sea necesario hasta lograr evitar el obstáculo completamente y poder seguir la pared.

Simulador ExaSim

En la Figura 45 puede verse la arquitectura de subsunción del comportamiento complejo para seguir la pared y entrar en la primera puerta evitando los obstáculos que puedan presentarse.

Como podemos ver en la Figura 45, el comportamiento complejo comienza por el comportamiento simple *seguir-pared* que se corresponde al esquema de la Figura 39 ya resuelto en el Punto 7.1.3. Se asume que el robot se encuentra originalmente cerca de la pared a seguir ubicada a izquierda. Luego el robot avanza siguiendo la pared hasta que encuentra un obstáculo.

Por esto mismo, se define una transición con nodo origen-*seguir-pared* y destino-*girar-derecha*. La condición que se debe cumplir para que se ejecute la transición será que el sensor frontal, *telemetro.0*, tenga un valor mayor a 50 %, como muestra la Figura 46.

El comportamiento *girar-derecha* consiste en girar el robot sobre el lugar a

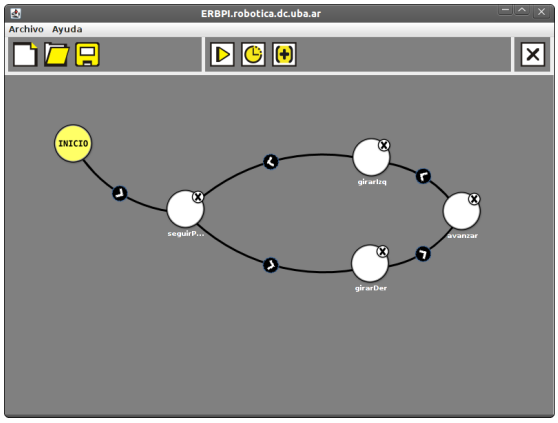

Figura 45: Esquema de comportamiento complejo para seguir la pared y entrar en la primera puerta, evitando los obstáculos que puedan presentarse, con el simulador ExaSim, compuesto de cuatro comportamientos simples *seguir-pared*, *girar-derecha*, *avanzar* y *girar-izquierda*.

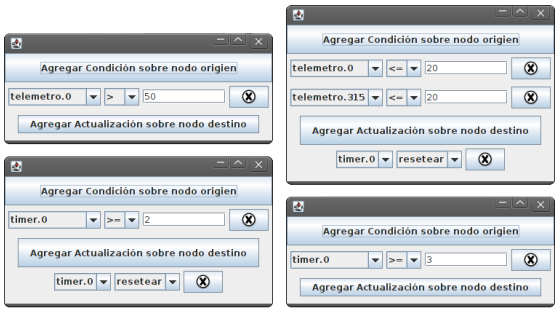

Figura 46: Ejemplo de configuración de condiciones para la transiciones del comportamiento complejo seguir la pared y entrar en la primera puerta, evitando los obstáculos que puedan presentarse, con el simulador ExaSim de la Figura 45. De izquierda a derecha y de arriba a abajo, las condiciones y actualizaciones de la transición con nodo origen-*seguir-pared* y destino-*girar-derecha*; la transición con nodo origen-*girar-derecha* y destino-*avanzar*; la transición con nodo origen-*avanzar* y destino-*girar-izquierda*; y la transición con nodo origen-*girar-izquierda* y destino-*seguir-pared*.

la derecha hasta que el robot detecte que ya puede avanzar sin tocar el obstáculo. Para esto, se define una nueva transición con nodo origen-*girar-derecha* y destino-*avanzar*. La condición que se debe cumplir para que se ejecute la transición es que los sensores frontal y frontal izquierdo, *telemetro.0* y *telemetro.315*, tengan un valor menor o igual a 20 %, como muestra la Figura 46.

El comportamiento simple *avanzar* consiste en avanzar en forma recta el robot un tiempo suficiente para que éste quede alejado del obstáculo. Para ello se define un temporizador *timer.0* que se reseteará en la transición entre *girar-derecha* y *avanzar*, como puede verse en la Figura 46. En este caso, se fijó experimentalmente el tiempo en dos segundos. Por esto mismo, se define una transición con nodo origen-*avanzar* y destino-*girar-izquierda* cuya condición es que el temporizador *timer.0* sea mayor o igual a dos. En la Figura 46 pueden verse las condiciones de esta transición.

De manera análoga, se resuelve el comportamiento *girar-izquierda* como girar el robot sobre el lugar en sentido a la izquierda el tiempo suficiente para que éste haya rodeado el obstáculo y quede nuevamente cerca de la pared. Nuevamente, en la transición entre *avanzar* y *girar-izquierda* se resetea el temporizador *timer.0* para que éste comience en cero al ejecutar *girar-izquierda*. Esta actualización también puede verse en la Figura 46. El comportamiento *girar-izquierda* se ejecuta durante tres segundos antes de volver la ejecución a *seguir-pared*. Por lo tanto, se define una nueva transición con nodo origen-*girar-izquierda* y destino-*seguir-pared* cuya condición es que el temporizador *timer.0* sea mayor o igual a tres. En la Figura 46 pueden verse las condiciones de esta transición.

Finalmente, como dijimos anteriormente, el comportamiento complejo resulta de la coordinación de estos cuatro comportamientos simples y queda definido como un ciclo iterativo de *seguir-pared*, *girar-derecha*, *avanzar* y *girar-izquierda* que se ejecutan consecutivamente las veces que sea necesario como puede verse en la Figura 45.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento complejo para seguir la pared y entrar en la primera puerta, evitando los obstáculos que puedan presentarse, con el simulador ExaSim.

Robot ExaBot

El mismo esquema de subsunción y comportamientos simples realizado para el ExaSim puede establecerse para el ExaBot, de forma que sólo cambia la plataforma sobre la que ejecutaremos el comportamiento. El esquema de conexiones para el ExaBot se corresponde con los de las Figuras 45 y 46.

En el material digital adjunto a este trabajo en el Apéndice A.2, pueden encontrarse videos de ejecuciones de este comportamiento complejo para seguir la pared y entrar en la primera puerta, evitando los obstáculos que puedan presentarse, con el robot ExaBot.

7.2.4. Seguir la pared desconociendo la ubicación inicial de la misma y entrar en la primera puerta

Puede agregarse al problema de seguir la pared la condición de que no se conoce la posición inicial del robot y, por lo tanto, en qué sentido se debe mover el robot inicialmente para poder encontrar la pared y seguirla. Nuevamente, es posible identificar en este comportamiento complejo, a dos comportamientos simples: *buscar la pared* y *seguir la pared*.

Salvando las diferencias entre una línea en el piso y una pared, el comportamiento simple *buscar la pared* no es otro que el comportamiento simple *buscar la línea* utilizado en la resolución del problema del Punto 7.2.1, donde se deberían utilizar los sensores para medir distancias en lugar de los de detección de línea.

De esta forma, este comportamiento complejo, se reduce a lograr una adecuada coordinación entre los comportamientos simples ya resueltos en el Punto 7.2.1 (*buscar la pared/línea*) y el Punto 7.1.3 (*seguir la pared*).

Otra posibilidad, incluso más compleja, sería agregar a este problema la condición de que también pueden aparecer obstáculos en el camino. Nuevamente, se pueden resolver este problema reutilizando y coordinando adecuadamente los comportamientos simples *buscar la pared* y *seguir la pared*, mencionados en este mismo problema, más el comportamiento simple del Punto 7.1.1 (*evitar obstáculo*), o su versión compuesta de tres comportamientos simples del punto anterior.

7.3. El ERBPI en las experiencias de Robótica Educativa

Durante el transcurso del desarrollo de la nueva herramienta, el LRSE ha realizado diversas experiencias didácticas, cursos, talleres y charlas de Robótica Educativa. La realización de estas actividades durante el desarrollo del ERBPI, permitieron poner a prueba la herramienta y establecer mejoras y cambios sobre la base de la práctica e interacción de los alumnos con la misma. De esta forma, se pudieron poner a prueba las ventajas del modelo utilizado para la herramienta como enfoque didáctico y sus potencialidades en la enseñanza en el marco de los talleres de Robótica Educativa.

En particular, se utilizaron versiones preliminares del ERBPI en tres cursos diseñados para alumnos de escuelas medias [6] [7] [8]. Estos cursos estaban integrados por alrededor de veinte participantes. Dos consistieron en ocho encuentros de cuatro horas y uno en dos encuentros de 3 horas. En la Figura 47 pueden verse algunas imágenes de estos encuentros.

Los conocimientos de programación de los alumnos fue variado, algunos tenían ciertos conocimientos de lenguajes de programación imperativa como C, C++ y Pascal. Muy pocos alumnos tenían conocimientos previos de electrónica y todos los alumnos manejaban los conocimientos básicos de funciones matemáticas. Sin embargo, ninguno de los alumnos tenía experiencia previa con robots.

Los cursos utilizaron los conceptos básicos de la robótica basada en comportamientos, utilizando versiones preliminares del ERBPI que sólo proveían la

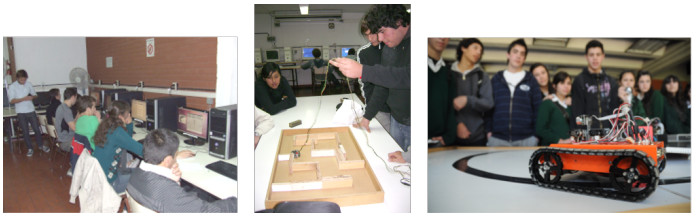

Figura 47: Talleres de Robótica Educativa para alumnos de escuelas medias realizados en 2009, 2010 y 2011.

interfaz para la programación de comportamientos simples siguiendo el modelo de los Vehículos de Braitenberg.

Siguiendo el enfoque didáctico para las experiencias de Robótica Educativa, desde el primer encuentro se formaron grupos de dos o tres participantes. Utilizando el enfoque de los Vehículos de Braitenberg, a los alumnos se les presentaron determinados problemas a resolver para los que desarrollaron comportamientos simples en los robots. En una primera instancia, resolvieron estos comportamientos en los simuladores Yaks y ExaSim. Luego, ejecutaron estos comportamientos en los robots reales Khepera y ExaBot. En la Figura 48 pueden verse algunas imágenes de estas experiencias.

Figura 48: Taller de Robótica para alumnos de escuelas medias.

En el transcurso de los encuentros, los alumnos debieron desarrollar distintos comportamientos de dificultad progresiva para resolver los problemas y objetivos planteados, desde seguir una fuente de luz, avanzar evitando obstáculos, seguir una línea en el piso, hasta resolver un laberinto. Frente a los problemas planteados a resolver, se incentivaba a los alumnos en el método científico, para lo cuál debían proponer hipótesis, contrastar los resultados esperados con los obtenidos, encontrar explicación a las diferencias con los resultados esperados y finalmente proponer cambios para ajustar los resultados obtenidos. Al mismo tiempo, cada grupo debía compartir al resto del taller sus resultados, explicando los diferentes enfoques y soluciones a los problemas planteados durante el desarrollo de la solución. Como resultado final y cierre del curso, los alumnos

debieron sintetizar en posters las actividades realizadas y sus experiencias, que luego fueron expuestos en una muestra final.

En su gran mayoría, los alumnos comenzaron a utilizar el ERBPI con facilidad, programando todos los comportamientos propuestos desde el primer encuentro de forma rápida.

Estas experiencias permitieron encontrar mejoras al ERBPI. La más importante de estas mejoras, frente al requerimiento de los alumnos de poder desarrollar comportamientos más complejos, fue la necesidad de una interfaz que permitiera la combinación de los distintos comportamientos simples basados en el modelo de los Vehículos de Braitenberg. De esta forma, surge el modelo de la arquitectura de subsunción basada en comportamientos como una solución a este problema.

Otra mejora importante surgida de estas experiencias fue la necesidad de que el ERBPI pudiera ejecutar el comportamiento directamente en el robot. En la versión del ERBPI utilizada en los cursos, la computadora donde el alumno trabajaba debía comunicarse permanentemente con el robot para realizar la transmisión de datos de sensores, actuadores y comandos. Esto disminuía significativamente la velocidad de respuesta del robot frente a cambios en el ambiente y, por lo tanto, la posibilidad de contar con un comportamiento verdaderamente reactivo. Por eso, se modificó el ERBPI para que pueda ejecutar el comportamiento localmente en el robot, en el caso que el robot con el que se está trabajando lo permita.

Las experiencias permitieron además realizar muchos cambios en la interfaz gráfica con el usuario de modo de facilitar y mejorar la interacción de la herramienta con los alumnos.

Al finalizar los talleres se realizaron encuestas a los alumnos en base a las encuestas de talleres de la Dirección de Orientación Vocacional de la FCEyN-UBA donde los alumnos detallaron su experiencia en el curso y con el uso del ERBPI. Los alumnos respondieron en su mayoría que habían encontrado al ERBPI fácil de aprender y utilizar a pesar de no tener conocimientos previos de programación y no haber controlado un robot antes del curso. También, todos los alumnos consideraron que el curso había cubierto sus expectativas y la mayoría de ellos mostraron interés en tomar más cursos de robótica, ciencias de la computación e ingeniería.

Por último, el ERBPI también fue utilizado en otras actividades como cursos de poca duración, conferencias, exposiciones y actividades de divulgación de ciencia para alumnos de escuelas medias y público en general. Entre estas actividades podemos citar el *Stand de Robótica en la Plaza de la Ciencias - ExpoUBA 2010* [52] en la que participaron más de 600 colegios con aproximadamente 70.000 visitas. La *Feria Nacional de Ciencias en Tecnópolis* en el marco del Concurso Nacional de Innovaciones - INNOVAR 2011 [53]. Charlas y talleres de robótica realizados en el marco de la *Semana de la Computación* en la FCEyN-UBA [54] [55] [56] [57] [58]. También, muchas de las ideas que fueron dando origen al desarrollo del ERBPI se originaron como resultado de experiencias anteriores en diversas charlas y talleres con alumnos de escuelas medias [59] [60] [61] [62]. En la Figura 49 pueden verse algunas imágenes de estas actividades.

Figura 49: Actividades de divulgación y exposiciones de ciencias para alumnos de escuelas medias y público en general realizados en 2009, 2010 y 2011.

8. Conclusiones y trabajos futuros

En este trabajo presentamos el ERBPI (Easy Robot Behaviour Programming Interface), una nueva interfaz gráfica de programación de robots móviles, fácil de aprender y usar, pensada especialmente para desarrollar actividades de Robótica Educativa para alumnos de escuela media. El ERBPI no requiere del usuario ninguna experiencia previa en programación ni robótica. Para lograr esto, abandonamos el paradigma de programación imperativa y adoptamos un enfoque basado en comportamientos. De esta forma, la nueva aplicación hace uso del paradigma conexionista, donde los comportamientos se definen estableciendo conexiones configurables entre los sensores y actuadores del robot. Por otra parte, los distintos comportamientos se pueden conectar entre sí mediante una arquitectura de subsunción. Esta arquitectura se representa mediante una máquina de estados finitos, donde cada estado representa un comportamiento y cada transición un cambio de comportamiento, que se activa cuando se cumplen determinadas condiciones asociadas a las acciones y percepciones del robot o a ciertas condiciones de contexto.

El ERBPI está diseñado para trabajar con diferentes tipos de robots y simuladores, abstrayendo los distintos sensores, actuadores, comandos y protocolos de comunicación que posee cada plataforma, haciendo posible añadir fácilmente nuevos robots o simuladores.

Las experiencias didácticas realizadas en distintos cursos y talleres con alumnos de la escuela media nos han permitido poner a prueba el ERBPI como una valiosa herramienta didáctica para la Robótica Educativa.

Presentamos además una formalización de la aplicación, que permite definir los alcances y límites desde el punto de vista del poder de expresividad del lenguaje que resulta al programar comportamientos con el ERBPI. La equivalencia demostrada con una AFSM puede permitir deducir propiedades sobre el ERBPI, a partir del conocimiento existente sobre estas máquinas de estados.

Como trabajo futuro nos proponemos aumentar la cantidad de funciones matemáticas que la aplicación provee, lo que ampliaría significativamente la capacidad de programación de comportamientos del ERBPI y, al mismo tiempo, la transformaría en una herramienta más atractiva para la ensenãnza de la matemática aplicada. Además nos proponemos incorporar nuevos robots para que puedan ser controlados con el ERBPI. En particular, el LRSE ha incorporado en el último tiempo un nuevo robot volador. Se trata de un cuadracóptero, más comúnmente denominado UAV (Unmanned Aerial Vehicle). Trabajaremos para desarrollar la RAL correspondiente y modificar el GUI para que la interfaz esté en condiciones de manejar este tipo de robots móviles no terrestres.

A. Apéndice

A.1. Archivos fuentes del ERBPI

En el DVD adjunto a este trabajo, en la carpeta `ERBPI-src`, se encuentran los archivos fuentes del ERBPI. Los mismos, están organizados de la siguiente manera:

- `ERBPI-src/core`
- `ERBPI-src/gui`
- `ERBPI-src/ral`

A.2. Registro de los experimentos

En el DVD adjunto a este trabajo, en la carpeta `ERBPI-experimentos`, se encuentra un registro de los experimentos realizados en el Capítulo 7. Los mismos se encuentran ordenados en carpetas según los problemas resueltos. En cada uno, pueden encontrarse los archivos de comportamientos utilizados y videos de las ejecuciones de los mismos. La organización de los experimentos es la siguiente:

- `avanzar_evitando_obstaculos`
- `seguir_linea`
- `seguir_linea_desconociendo_ubicacion_inicial`
- `seguir_pared_y_entrar_en_puerta`
- `seguir_pared_y_entrar_en_puerta_evitando_obstaculos`

Referencias

[1] Velasco R. *Ciencia y Tecnología a través de la Robótica Educativa*. Perfiles Educativos, Universidad Nacional Autónoma de México, 1996.

[2] Patiño K., Diego B., Moreno V. *Experiencias con Robótica Educativa en el Centro Internacional de Tecnologías Avanzadas*. Teoría de la Educación y Cultura en la Sociedad de la Información, Universidad de Salamanca, 2011.

[3] Chavarría M., Saldaño A. *La Robótica Educativa como una innovativa interfaz educativa entre el alumno y una situación-problema*. Didáctica y Educación. ISSN 2224-2643, Vol 1, No 2, 2010.

[4] Alimisis D., Arlegui J., Fava N., Frangou S., Ionita S., Menegatti E., Monfalcon S., Moro M., Papanikolaou K., Pina A. *Introducing robotics to teachers and schools: experiences from the TERECoP project*. Proceedings of the Constructionism 2010 Conference, Francia, 2010.

[5] Detsikas N., Alimisis D. *Status and Trends in Educational Robotics Worldwide*. School of Pedagogical and Technological Education, Grecia, 2011.

[6] *Taller de Programación de Robots*. Talleres de Investigación y Tecnología en Computación para estudiantes de escuela media, Dirección de Orientación Vocacional, FCEyN-UBA, 2009.

[7] *Taller de Robótica para estudiantes de colegios secundarios*. Laboratorio de Robótica y Sistemas Embebidos, Departamento de Computación, FCEyN-UBA, 2010.

[8] *Taller de Programación de Robots*. Talleres de Investigación y Tecnología en Computación para estudiantes de escuela media, Dirección de Orientación Vocacional, FCEyN-UBA, 2011.

[9] Braitenberg V. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, 1986.

[10] Pedre S., De Cristóforis P., Caccavelli J., Stoliar A. *ExaBot: A mobile mini robot architecture for research, education and popularization of science*. Journal of Applied Computer Science Methods, Guest Editors: Jacek Zurada and Pablo Estevez, vol 2, No 1 pp 41-59, ISSN 1689-9636, 2010.

[11] Mondada F., Franzi E., Guilgnard A., Nicoud J. *Khepera: The Optimal Tool for Development in Mobile Robotics*. IEEE International Conference on Robotics and Automation, ISSN 1050-4729, 1995.

[12] Vaughan R., Gerkey B. *Really Reusable Robot Code and the Player/Stage Project*. Software Engineering for Experimental Robotics, Springer Tracts on Advanced Robotics, D. Brugali, pp. 267-289, 2007.

[13] Carlsson J., Ziemke T. *YAKS - Yet Another Khepera Simulator*. 1st International Symposium on Autonomous Minirobots for Research and Edutainment - AMiRE2001, Proceedings of the 5th International Heinz Nixdorf Symposium (pp. 235-241), HNI Verlagsschriftenreihe, vol. 97, Heinz Nixdorf Institute, Alemania, 2001.

[14] Blank D., Kumar D., Meeden L., Yanco H. *Pyro: A python-based versatile programming environment for teaching robotics*. Journal on Educational Resources in Computing - JERIC, Special issue on robotics in undergraduate education, part 2, 4(3):115, 2004.

[15] Baum D. *NQC: Not Quite C*. `http://bricxcc.sourceforge.net/nqc`.

[16] Markus N. *BrickOS*. `http://brickos.sourceforge.net`.

[17] Solorzano J. *leJOS: Java for Lego Mindstorms*. `http://lejos.sourceforge.net`.

[18] Tufts University. *RoboLab*. `http://www.ceeo.tufts.edu/robolabatceeo`.

[19] *StarLogo TNG: The Next Generation*. MIT's Scheller Teacher Education Program - STEP. `http://education.mit.edu/drupal/starlogo-tng`.

[20] *Squeak EToys*. `http://www.squeakland.org`.

[21] Resnick M., Maloney J., Monroy-Hernández A., Rusk N., Eastmond E., Brennan, K., Millner M., Rosenbaum E., Silver J., Silverman S., Kafai Y. *Scratch: Programming for All*. Communications of the ACM, vol. 52, No. 11, 2009.

[22] Arkin R. C. *Behavior-Based Robotics*. MIT Press, Cambridge, 1998.

[23] Catlin D., Blamires M. *The Principles of Educational Robotic Applications: A framework for understanding and developing educational robots and their activities.* Proceedings for Constructionism 2010, 12th EuroLogo Conference, Paris, France, 2010.

[24] Arlegui J., Fava N., Menegatti E., Monfalcon S., Moro M., Pina A. *Robotics at primary and secondary education levels: technology, methodology, curriculum and science.* Proceedings of the 3rd International Conference ISSEP Informatics in Secondary Schools Evolution and Perspectives, Torun, Polonia, 2008.

[25] Caccavelli J., Pedre S., De Cristóforis P., Stoliar A., Katz A., Bendersky D. *Desarrollo de una interfaz de programación para talleres de Robótica Educativa.* Proyecto de Extensión "Exactas con la Sociedad 2008-2009", Laboratorio de Robótica y Sistemas Embebidos, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2010.

[26] Brooks R. *A Robust Programming Scheme for a Mobile Robot.* Proceedings of NATO Advanced Research Workshop on Languages for Sensor-Based Control in Robotics, Castelvecchio Pascoli, Italy, 1986.

[27] Brooks R. *A Robust Layered Control System for a Mobile Robot.* IEEE Journal of Robotics and Automation, Vol. 2, No. 1, pp. 14-23, 1986.

[28] Brooks R. *Elephants Don't Play Chess - Designing Autonomous Agents.* MIT Press, Cambridge MA, 1990.

[29] Arkin R. C. *Towards the Unification of Navigational Planning and Reactive Control.* Working Notes of the AAAI Spring Symposium on Robot Navigation, Stanford University, 1999.

[30] De Cristóforis P., Pedre S., Caccavelli J., Stoliar A. *ExaBot: a mini robot for research, education and popularization of science.* VI Escuela de Verano Latino-americana en Inteligencia Computacional y Robótica, Chile, 2009.

[31] Gerkey B., Vaughan R., Howard A. *The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems.* Proceedings of the 11th International Conference on Advanced Robotics, pp. 317-323, Portugal, 2003.

[32] Mondada F., Franzi E., Ienne P. *Mobile Robot Miniaturisation: A tool for investigation in control algorithms.* Proceedings of the 3rd International Symposium on Experimental Robotics, Springer Verlag, pp 501-513, 1994.

[33] Papert S., Watt D., di Sessa A., Weir, S. *Final report of the Brookline Logo Project: An assessment and documentation of a children's computer laboratory.* MIT Division for Study and Research in Education, Massachusetts Institute of Technology, 1979.

[34] Gaudiello I., Zibetti E., Carrignon S. *Representations to go: learning robotics, learning by robotics.* Proceedings of SIMPAR 2010 Workshops, International Conference on Simulation, Modeling and Programming for Autonomous Robots, ISBN 978-3-00-032863-3, pp. 484-493, Alemania, 2010.

[35] Monsalves S. *Estudio sobre la utilidad de la robótica educativa desde la perspectiva del docente.* Revista de Pedagogía, Red de Revistas Científicas de América Latina, el Caribe, España y Portugal, Vol. 32, No. 90, pp. 81-117, 2011.

[36] Frangou S., Papanikolaou K., Aravecchia L., Montel L., Ionita S., Arlegui J., Pina A., Menegatti E., Moro M., Fava N., Monfalcon S., Pagello I. *Representative examples of implementing educational robotics in school based on the constructivist approach.* Proceedings of SIMPAR 2008 Workshops, International Conference on Simulation, Modeling and Programming for Autonomous Robots, ISBN 978-88-95872-01-8, pp. 54-65, Italy, 2008.

[37] Biggs G., MacDonald B. *A Survey of Robot Programming Systems.* Proceedings of the Australian Conference on Robotics and Automation, Australia, 2003.

[38] Kramer J., Scheutz M. *Development environments for autonomous mobile robots: A survey.* Journal of Autonomous Robots, Springer, No. 22, pp. 101-132, 2007.

[39] Mohamed N., Al-Jaroodi J., Jawhar I. *Middleware for Robotics: A Survey.* Proceedings of the IEEE International Conference on Robotics, Automation and Mechatronics, pp. 736-742, 2008.

[40] *Microsoft Robotics Developer Studio.* `http://www.microsoft.com/robotics`.

[41] Azhar M., Goldman R., Sklar E. *An Agent-oriented Behavior-based Interface Framework for Educational Robotics.* Agent-Based Systems for Human Learning - ABSHL, Workshop at Autonomous Agents and MultiAgent Systems, AAMAS, 2006.

[42] Alexander G., Heckel F., Youngblood G., Hale D., Ketkar N. *Rapid Development of Intelligent Agents in First/third-person Training Simulations via Behavior-based Control.* Proceedings of the 19th Conference on Behavior Representation in Modeling and Simulation, 2010.

[43] Qiu F., Hu X. *BehaviorSim: A Learning Environment for Behavior-based Agent.* Proceedings of the 10th International Conference on the Simulation of Adaptaive Behavior - SAB, 2008.

[44] Reckhaus M., Hochgeschwender N., Ploeger P., Kraetzschmar G. *A Platform-independent Programming Environment for Robot Control.* 1st International Workshop on Domain-Specific Languages and models for Robotic Systems - DSLRob, 2010.

[45] Tousignant S., Van Wyk E., Gini M. *An Overview of XRobots: A Hierarchical State Machine Based Language.* Department of Computer Science and Engineering, University of Minnesota, 2011.

[46] *XML: Extensible Markup Language.* `http://www.w3.org/XML`.

[47] *DOM: Document Object Model.* `http://www.w3.org/DOM`.

[48] Cormen H., Leiserson C., Rivest R., Stein C. *Introduction to Algorithms.* MIT Press, Cambridge, 2009.

[49] *C++ Standard Template Library.* `http://www.sgi.com/tech/stl`.

[50] Hopcroft J., Ullman J. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Series in Computer Science, 1979.

[51] *IX Competencia Nacional de Robótica.* Bahía Blanca, Buenos Aires, Argentina, 2011. `http://grsbahiablanca.com.ar/compe_2011.htm`.

[52] *Stand de Robótica en la Plaza de la Ciencias.* ExpoUBA 2010, Plaza de las Ciencias, Universidad de Buenos Aires, 2010.

[53] *INNOVAR - Concurso Nacional de Innovaciones.* Tecnópolis, Feria Nacional de Ciencias, Ministerio de Ciencia, Tecnología e Innovación Productiva de la Argentina, 2011.

[54] *Taller de Programación de Robots.* Semana de la Computación, Departamento de Computación y Secretaría de Extensión, Graduados y Bienestar, FCEyN-UBA, 2009.

[55] *Robótica en Exactas, Experiencia Demostrativa.* Semana de la Computación, Departamento de Computación y Secretaría de Extensión, Graduados y Bienestar, FCEyN-UBA, 2010.

[56] *Taller de Robótica.* Semana de la Computación, Departamento de Computación y Secretaría de Extensión, Graduados y Bienestar, FCEyN-UBA, 2010.

[57] *¿Cómo se hace un robot?.* Semana de la Computación, Departamento de Computación y Secretaría de Extensión, Graduados y Bienestar, FCEyN-UBA, 2011.

[58] *Sumo de Robots.* Semana de la Computación, Departamento de Computación y Secretaría de Extensión, Graduados y Bienestar, FCEyN-UBA, 2011.

[59] *Taller de Programación de Robots para alumnos de escuela secundaria.* Talleres de Investigación y Tecnología en Computación para estudiantes de escuela media, Dirección de Orientación Vocacional, FCEyN-UBA, 2006.

[60] *Taller para estudiantes del CBC de Computación.* Programa de Tutorias de la Secretaría de Extensión, Graduados y Bienestar, FCEyN-UBA, 2009.

[61] *Charlas para estudiantes del último año del secundario.* Visitas de estudiantes a los laboratorios del Departamento de Computación, Dirección de Orientación Vocacional, FCEyN-UBA, 2009.

[62] *Programando Robots: ¿Cuánto necesito saber?.* Charla en el marco del 12vo Torneo de Computación y Matemática 2009 para alumnos de escuelas secundarias, Departamento de Computación, FCEyN-UBA, 2009.