Nuevo Módulo de Conexión Inalámbrico vía Bluetooth con Arduino en la Librería JFML

Francisco Jesús Arcos
Dpt. de Ciencias de la Computación e I.A.
Universidad de Granada
18071 Granada, España
Email: pacoarcos82@gmail.com

José Manuel Soto-Hidalgo
Dpt. de Electrónica y Tecnología Electrónica
Universidad de Córdoba
14071 Córdoba, España
Email: jmsoto@uco.es

Autilia Vitiello
Dpt. de Ciencias de la Computación
Universidad de Salerno
84084 Salerno, Italia
Email: avitiello@unisa.it

Giovanni Acampora
Dpt. de Física Ettore Pancini
Universidad de Nápoles Federico II
80126 Nápoles, Italia
Email: giovanni.acampora@unina.it

Jesús Alcalá-Fdez
Instituto de Investigación DaSCI
Universidad de Granada
18071 Granada, España
Email: jalcala@decsai.ugr.es

***Resumen*—Los Sistemas Basados en Reglas Difusas han sido aplicados con éxito en un amplio rango de aplicaciones reales. Recientemente, la IEEE publicó el estándar IEEE Std 1855^{TM}-2016 para proporcionar a la comunidad una herramienta única y bien definida que permita diseñar sistemas completamente independiente del hardware/software específico. La librería Java Fuzzy Markup Language es la primera herramienta software de código abierto que ofrece una implementación completa de dicho estándar. Sin embargo, la versión actual de esta librería no permite conectarse a sistemas embebidos, lo que dificulta la aplicación de los sistemas difusos a problemas de control. El objetivo de este trabajo es desarrollar un nuevo módulo que permita diseñar y ejecutar un controlador difuso en el sistema embebido Arduino vía Bluetooth. Para ilustrar el potencial de este módulo se ha desarrollado un caso de uso en el que se utiliza un controlador difuso para manejar un robot movil para el comportamiento de seguimiento de contornos.**

***Index Terms*—Sistemas Basados en Reglas Difusas, JFML, Arduino, Software de Código Abierto, Hardware Libre**

I. Introducción

Los Sistemas Basados en Reglas Difusas (SBRD) [1] han sido aplicados con éxito en multiples campos, tales como clasificación [2], regresión [3], control [4], etc., debido a su capacidad para incluir conocimiento experto a priori, manejar la imprecisión existente en los datos, y representar sistemas para los que no es posible obtener un modelo matemático por la tipología del problema y de los datos. En concreto, los controladores difusos son un modelo especifico de SBRDs que permite controlar sistemas complejos de una manera similar a como lo hace un humano.

Para poder hacer uso de estos controladores en muchos problemas reales es necesario que puedan ser ejecutados en sistemas embebidos y conectados con un conjunto de sensores/actuadores. Los sistemas embebidos son dispositivos integrados diseñados para realizar una o unas pocas funciones específicas. En este escenario, es crucial poder integrar SBRDs en sistemas embebidos específicos.

Recientemente, la IEEE Computational Intelligence Society (IEEE-CIS) ha patrocinado la publicación del nuevo estándar para SBRD (IEEE Std 1855^{TM}-2016) [5] en el que se define el lenguaje Fuzzy Markup Language (FML) [6] basado en W3C eXtensible Markup Language (XML). El objetivo de este estándar es proporcionar a la comunidad una herramienta única y bien definida que permite un diseño del sistema completamente independiente del hardware/software específico.

Para que el estándar IEEE sea operativo y útil para la comunidad, se ha desarrollado la librería Java Fuzzy Markup Language (JFML)[1] de código abierto que ofrece una implementación completa del estándar. Sin embargo, la versión actual de la librería JFML no permite conectarse a dispositivos embebidos, lo que dificulta la aplicación de los SBRDs diseñados a problemas reales (centralizados o distribuidos). Aunque algunas librerías permiten utilizar SBRDs en algunos dispositivos hardware, tales como Matlab para Arduino[2] o eFLL[3], actualmente no existe una infraestructura software de código abierto que soporte SBRD diseñados según el estándar IEEE 1855^{TM}-2016 [7]. Además, en general, estas implementaciones se basan en aproximaciones reducidas de la lógica difusa que se ejecutan en el microcontrolador del dispositivo embebido, limitando su capacidad de cálculo.

El objetivo principal de este trabajo es desarrollar un módulo de código abierto para la librería JFML que permita diseñar y ejecutar SBRD en Arduino [8], uno de los sistemas embebidos más conocidos y utilizados en el modelo de hardware libre [9]. Con este módulo, los sensores/actuadores pueden ser asignados a las variables lingüísticas de la base de conocimiento permitiendo un diseño completamente independiente de las

Este trabajo esta soportado por el Ministerio de Economía y Competitividad bajo el proyecto TIN2014-57251-P y el Ministerio de Economía, Industria y Competitividad TIN2017-89517-P.

[1]https://github.com/sotillo19/JFML
[2]http://playground.arduino.cc/Interfacing/Matlab
[3]https://github.com/zerokol/eFLL

restricciones específicas de hardware/software que caracterizan la arquitectura dada en la cual se despliega el sistema. Además, se ha definido un nuevo protocolo de comunicación vía Bluetooth entre JFML y Arduino que permite realizar las inferencias en el ordenador en que se encuentra instalado el núcleo de JFML, eliminando así la limitada capacidad de computación que ofrecen los sistemas embebidos. Para ilustrar el potencial de este módulo mostramos un caso de uso con un controlador difuso para manejar un robot móvil en el comportamiento de seguimiento de contornos. En particular, mostramos cómo el nuevo módulo de JFML puede usarse para implementar SBRDs en Arduino de una manera fácil y lista para ser ejecutada sin ninguna programación adicional de Arduino. Como resultado, el nuevo módulo permite la interoperabilidad total en el diseño de SBRDs con placas Arduino.

Este trabajo está organizado de la siguiente manera. La sección II presenta una breve introducción del sistema embebido Arduino e introduce la librería JFML y sus principales características. La sección III describe el nuevo módulo de JFML para sistema embebido Arduino. La sección IV muestra un caso de uso centrado en cómo se puede utilizar el nuevo módulo para ejecutar un controlador difuso para el seguimiento de contornos de un robot móvil. Por último, en la sección V se indican algunas observaciones finales y trabajos futuros.

II. Preliminares

Esta sección pone en contexto los SBRDs, introduce el sistema embebido Arduino así como la librería Java de código abierto JFML.

II-A. Sistemas Basados en Reglas Difusas (SBRDs)

La mayoría de las aplicaciones reales en el ámbito de sistemas control tienen en común diversas características que hacen difícil el modelado del sistema utilizando las estrategias clásicas de modelado. Por ejemplo, en sistemas donde los procesos se describen de forma imprecisa sin recurrir a modelos matemáticos o algoritmos de los procesos físicos involucrados. En estos casos, la lógica difusa representa una poderosa herramienta con la que poder diseñar modelos precisos en sistemas complejos donde, debido a la complejidad o la imprecisión, las herramientas clásicas no tienen éxito.

En áreas de investigación como economía, procesamiento de imágenes, etc., se han aplicado con éxito los sistemas difusos [10], [11]. No obstante, posiblemente el control difuso sea el ámbito de aplicación más común de la lógica difusa donde un gran número de publicaciones lo avalan [12]–[14]. Estos sistemas, llamados SBRDs, constan de: una Base de Conocimiento (BC) que contiene las particiones difusas y el conjunto de reglas que modelan el sistema; una interfaz de Fuzzificación que transforma los valores entrada nítidos en conjuntos difusos; un sistema de Inferencia Difusa que realiza el razonamiento difuso haciendo uso de los valores de entrada y de la BC; y la interfaz de Defuzzificación que transforma la salida difusa de la inferencia en valores nítidos.

II-B. Sistemas embebidos: Arduino

Un sistema embebido es una pieza de hardware fuertemente especializada para un propósito específico que es embebido como parte de un dispositivo completo [15]. Dado que se dedican a tareas específicas, los ingenieros pueden optimizarlas para reducir el tamaño y el coste del producto y aumentar la fiabilidad y el rendimiento. Debido a esto, en los últimos años la industria del hardware ha experimentado un crecimiento exponencial y los sistemas embebidos controlan muchos de los dispositivos de uso común en la actualidad. Se estima que el 98 % de todos los microprocesadores se fabrican como parte de un sistema embebido.

Uno de los sistemas embebidos más conocidos y utilizados en el modelo de hardware libre es Arduino [8]. Es una tarjeta microcontroladora que contiene un conjunto de pines de entrada/salida digitales y analógicos que permiten conectarla a varias tarjetas de expansión y componentes electrónicos (antenas Bluetooth o Wifi, sensores, actuadores, entre otros), y su microcontrolador se puede programar utilizando un dialecto de los lenguajes de programación C y C++ (archivos con extensión *.ino*). Debido a su gran simplicidad y usabilidad es una herramienta popular para el desarrollo de productos de Internet of Things, así como una de las herramientas más exitosas para la educación STEM/STEAM[4]. Por ejemplo, en [16] se propuso el diseño de un robot móvil de bajo coste basado en Arduino como herramienta educativa alternativa o complementaria en laboratorios, aulas, e-learning y cursos abiertos masivos online (MOOC); Khater et al. [17] presentaron un controlador difuso adaptativo basado en Arduino DUE para gestionar un motor de corriente continua con eje flexible; etc.

II-C. Librería Java Fuzzy Markup Language

En el año 2000, el Fuzzy Control Language (FCL) fue definido en la norma IEC61131-7 de la Comisión Electrotécnica Internacional [18] con el objetivo de proporcionar a los ingenieros con una herramienta común y bien definida para integrar SBRDs en problemas de control, facilitando el intercambio de controladores entre diferentes lenguajes de programación y software (aumentando la usabilidad del software disponible). Recientemente, la IEEE-CIS patrocinó la publicación del estándar IEEE Std 1855^{TM}-2016 [19] en el que definen el lenguaje FML basado en W3C XML para extender estas ventajas a otros tipos de problemas (por ejemplo, clasificación o regresión), y para hacer uso de las ventajas XML para representar SBRDs.

JFML es una librería Java de código abierto (Licencia Pública GNU GPLv3)[5] para diseñar y usar SBRDs de acuerdo al estándar IEEE 1855^{TM}-2016. Esta librería incluye los 4 sistemas de inferencia (Mamdani, TSK, Tsukamoto y AnYa) y todas las funciones de pertenencia (MFs), operadores difusos, etc., considerados en la definición de esquema XML (XSD) del estándar (ver [19] para más información). JMFL incluye métodos personalizados (nombrados según el patrón

[4]http://stemtosteam.org/

[5]https://github.com/sotillo19/JFML

custom_name) para todos los elementos indicados en el XSD que permiten ampliar la librería de acuerdo con el estándar sin necesidad de modificar la gramática del lenguaje (por ejemplo, con MFs tipo 2 o intuitivo). Además, esta librería utiliza la misma estructura de árbol etiquetada que el estándar, lo que facilita la integración de cambios futuros del estándar. JFML hace uso de la nueva API de Java JAXB para proporcionar una forma rápida y conveniente de leer y escribir SBRDs de acuerdo al estándar. Además, para facilitar la interoperabilidad de la librería con otros software disponibles, JFML incluye un módulo para importar/exportar SBRDs desde documentos FCL, PMML y fis (sistemas diseñados con la Toolbox de lógica difusa de Matlab). Una amplia documentación sobre la librería junto con una buena variedad de ejemplos puede encontrarse en la página web [6].

III. Módulo para Arduino vía Bluetooth

En esta sección presentamos el nuevo módulo para conectar JFML con Arduino vía Bluetooth. La subsección III-A muestra una visión general del diagrama de clases principal del módulo. La subsección III-B describe el protocolo de comunicación vía Bluetooth para la comunicación bidireccional entre JFML y Arduino. Y finalmente, el diagrama de clases para el sistema embebido Arduino y el conjunto de sensores/actuadores incluidos en el módulo son presentados en la subsección III-C.

III-A. Diseño General

A continuación, se introducen las principales clases que compone el nuevo módulo con el objetivo de proporcionar una visión general de las posibilidades que ofrecen.

1. *EmbeddedSystem*: Es la clase principal responsable de definir las características y el tipo de conexión con Arduino. Requiere un nombre, un puerto serie y una velocidad de transmisión para establecer la conexión con el núcleo JFML. Tiene asociada una lista de objetos *EmbeddedVariable* que representa la relación entre las variables de la base de conocimiento y los sensores/actuadores. Además, esta clase define un método abstracto para crear un archivo ejecutable de acuerdo con los requisitos de diseño de la arquitectura de Arduino. Gracias a este archivo, un usuario puede ejecutar fácilmente un SBRD sin necesidad de tener conocimientos específicos sobre la arquitectura y el lenguaje de programación de Arduino.

2. *EmbeddedController*: Esta clase permite coordinar todas las placas Arduino conectadas a JFML. Para ello contiene una lista de objetos *EmbeddedSystem* y un objeto de la clase *FuzzyInferenceSystem* para determinar el tipo de inferencia difusa considerada. Además, esta clase permite al usuario indicar si el sistema debe funcionar durante un periodo de tiempo fijo o de forma indefinidamente.

3. *EmbeddedVariable*: Esta clase representa la relación entre los sensores/actuadores y las variables de la base de conocimiento, indicando para cada uno de ellos el puerto de entrada/salida asociado. Destacar que el dominio de los sensores/actuadores y de las variables son escalados para que haya una relación directa entre los dominios.

[6]http://www.uco.es/JFML

4. *Sensor* y *SensorLibrary*: Estas clases son las encargadas de definir las características de los diferentes sensores/actuadores, incluyendo: el tipo (analógico o digital), el dominio (valor máximo y mínimo), y una librería de operaciones especificas. Destacar que cada sensor/actuador tiene asociada una librería con los códigos de operaciones asociados al mismo (código de inicialización, etc.).

5. *AggregatedSensor*: Esta clase permite definir un nuevo tipo de sensor que genera información útil para el SBRD a partir de los valores proporcionados por uno o varios objetos de la clase *Sensor*. Destacar que esta clase también puede ser utilizada sin objetos de la clase *Sensor* para generar una entrada del controlador mediante una función, variable o constante que no requiera el valor de los sensores de entrada.

6. *Connection* y *ConnectionLibrary*: La clase *Connection* especifica un método de conexión bidireccional que permite conectar Arduino con el núcleo de JFML, definiendo un protocolo de comunicación entre ellos. La clase *ConnectionLibrary* permite incluir un código adicional para conectar las placas Arduino si es necesario. En la siguiente subsección se mostrará en detalle el protocolo de comunicación diseñado.

III-B. Protocolo de Comunicación

El nuevo módulo permite que la inferencia se realice en el ordenador en el que esté instalado el núcleo de JFML, evitando que se tenga que realizar en el sistema embebido que normalmente tiene menor capacidad computacional y de memoria. Para ello, se ha definido un nuevo protocolo de comunicación entre JFML y Arduino tanto para la conexión como para el intercambio de mensajes entre ambos. Este protocolo realiza la gestión de la comunicación mediante peticiones de mensajes de lectura/escritura que consta de los siguientes pasos:

Primero, el controlador (clase *EmbeddedController*) envía un mensaje de difusión a todas las placas Arduino para establecer la conexión con JFML. Cada Arduino recibe un mensaje de lectura y/o escritura de los valores de los sensores conectados en función de la relación establecida entre los sensores y las variables de la base de conocimiento. Por ejemplo, si un sensor está asociado a una variable de entrada, el sistema embebido obtiene un valor del sensor. Sin embargo, si el sensor/actuador se asocia a una variable de salida, el sistema embebido establece un valor para el sensor. Este valor se envía a JFML como un mensaje según la definición del protocolo.

Segundo, el controlador espera una respuesta (un mensaje con el valor de los sensores) de todas las placas Arduino conectadas. Si el controlador recibe respuesta de todos ellos y no se supera el tiempo de espera definido, los valores de las variables de la base de conocimiento se actualizan con los valores recibidos (procedentes de los sensores). En caso contrario se muestra un mensaje de error.

Tercero, JFML hace la inferencia difusa teniendo en cuenta los valores de las variables de entrada. Posteriormente, los valores de salida se envían a las placas de Arduino correspondientes (aquellas que tienen sensores asociados con variables

de salida). Los actuadores asociados a las variables de salida se ajustan de acuerdo con los resultados de la inferencia difusa. Finalmente, el controlador espera una respuesta ACK de todos los Arduinos indicando que los resultados de la inferencia difusa están correctamente configurados en los sensores o actuadores correspondientes, de lo contrario, aparece un mensaje de error.

III-C. Diseño de Clases para el Sistema Embebido Arduino

Se lleva a cabo una implementación basada en Arduino de acuerdo con el diagrama de clases introducido en la subsección III-A. Para ello se implementa una nueva clase *EmbeddedSystemArduino* que extiende a la clase abstracta *EmbeddedSystem*. Además, también se desarrolla una comunicación bidireccional por el puerto Bluetooth para que las placas Arduino pueden conectarse y usarse para diseñar SBRDs de forma sencilla.

La clase *EmbeddedSystemArduino* implementa el método abstracto para crear automáticamente un archivo ejecutable (*.ino*) de acuerdo con el diseño de la arquitectura de Arduino. Este archivo *.ino* incluirá el código necesario para iniciar y ejecutar el sistema en la placa Arduino. Requiere dos funciones: *setup()* y *loop()*. La función *setup()* es ejecutada cuando la placa Arduino es encendida o se ha utilizado la opción reset. Principalmente es utilizada para inicializar variables, los modos de los pines de entrada y salida, e inicializar otras librerías necesarias en el programa. La función *loop()* es ejecutada después de la función *setup()* dentro de un bucle en el programa principal. Está función incluye el código necesario para leer y actualizar constantemente el valor de los sensores y actuadores. El método abstracto incluye en el código de la función *setup()* las variable de la base de conocimiento que está asociada con un sensor o actuador (además de las bibliotecas asociadas), el modo de cada una (entrada o salida), y el número del pin al que está conectado. Además, este método incluye en la función*loop()* el código necesario mantener la comunicación entre JFML y la placa Arduino. El fichero *.ino* generado automáticamente debe ser cargado directamente en la ROM de Arduino mediante la IDE de la empresa. Destacar que este fichero puede ser modificado por el usuario para incluir su propio código si es necesario.

Este módulo también incluye una colección de los sensores y actuadores más utilizados en las propuestas con Arduino [20]. Además, los usuarios pueden añadir nuevos sensores y/o actuadores facilmente extendiendo la clase *Sensor*. El cuadro I muestra una lista con los 7 sensores y los 5 actuadores integrados en el módulo.

Cuadro I
Sensores y actuadores incluidos en el módulo

Sensor	**Tipo**	**Digital**	**Analógico**
DHT22	Temperatura y Humedad	X	
HC-SR04	Distancia (ultrasonido)	X	
HC-SR501	Movimiento	X	
LDR	Intensidad lumínica	X	X
MPU6050	Acelerómetro y Giroscopio	X	
MQ-2	Gas	X	X
H206	Encoder	X	
Actuador	**Tipo**	**Digital**	**Analógico**
DC Motor	Motor corriente continua	X	X
L298N	Driver	X	X
LED Color	Led	X	
LED PWM	Led	X	
SG90 Motor	Servo motor		X

IV. Caso de Uso

Con el objetivo de mostrar el potencial del nuevo módulo de JFML, se ha realizado un caso de uso con un SBRD para controlar un robot móvil para el comportamiento de seguimiento de contornos. En robótica móvil, los controladores difusos son comúnmente considerados para realizar este comportamiento que es muy utilizado para explorar estancias interiores desconocidos. El objetivo es mantener una distancia adecuada entre el robot y la pared (d_{wall}) mientras el robot se mueve lo más rápido posible, evitando cambios bruscos en los movimientos y en la velocidad. En [21], los autores diseñaron un controlador difuso para seguimiento de contornos para el robot Nomad-200 haciendo uso de algoritmos genéticos. Éste consta de cuatro variables de entrada: relación entre la distancia a la derecha y d_{wall} (RD), relación entre la distancia a la izquierda y de la derecha (DQ); orientación respecto a la pared (O); y velocidad lineal respecto a la velocidad lineal máxima del robot (V); y dos variables de salida: aceleración lineal (LA) y velocidad angular (AV) respecto a la aceleración lineal máxima y velocidad angular del robot, respectivamente. Este controlador difuso está disponible como uno de los ejemplos proporcionados con el software JFML ($./Ejemplos/XMLFiles/RobotMamdani.xml$).

Para este caso de uso montamos un robot móvil con componentes de bajo coste que consta de una placa Arduino MEGA 2560, cinco sensores de distancia HC-SR04, dos sensores de velocidad del motor H206, un sensor acelerómetro/giroscopio MPU6050, y dos motores de corriente continua con el conductor H-Bridge L298N (ver Figura 1). El robot móvil es controlado en tiempo real desde el ordenador portátil utilizando la librería JFML para ejecutar el controlador difuso y gestionar la comunicación con los sensores/actuadores vía Bluetooth. Destacar que los valores de los sensores no se utilizan directamente como entradas del controlador difuso. Varios objetos de la clase *Aggregated Sensor* son utilizados para generar los valores de entrada del controlador a partir de los valores obtenidos de distintos sensores. Asimismo, las salidas del controlador se envían a otros objetos de la clase *Aggregated Sensor* para generar los valores de los motores.

Para llevar a cabo todo el proceso, es necesario realizar los siguientes pasos. El primer paso es leer el controlador difuso desde el documento FML. El segundo paso es crear los sensores/actuadores que están conectados con la placa Arduino. El tercer paso es generar los objetos de la clase *Aggregated Sensor* para los valores de entrada y salida. El cuarto paso es mapear las variables de entrada/salida con los sensores/actuadores e incluirlas en una lista. El quinto paso es crear el sistema embebido (incluyendo el nombre, puerto, etc.) y generar el archivo *.ino*. Estos pasos implementan con

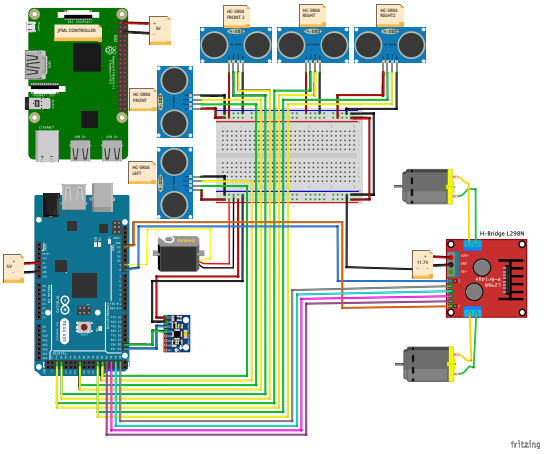

Figura 1. Arquitectura Hardware del Robot Movil.

siguiente código Java:

```
// Paso 1: Lectura del controlador desde un fichero FML
File fml = new File("./Examples/XMLFiles/RobotMamdani.xml");
FuzzyInferenceSystem fs = JFML.load(fml);

// Paso 2: Crear los sensores/actuatores
KnowledgeBaseVariable var_rd =  fs.getVariable("rd");
KnowledgeBaseVariable var_dq =  fs.getVariable("dq");
KnowledgeBaseVariable var_o =  fs.getVariable("o");
KnowledgeBaseVariable var_v =  fs.getVariable("v");
KnowledgeBaseVariable var_la =  fs.getVariable("la");
KnowledgeBaseVariable var_av =  fs.getVariable("av");

Sensor rd_front = new ArduinoHC_SR04(rd.getName()+"front",
ArduinoPin.PIN_40, ArduinoPin.PIN_41, 10, 200, true, 3,
true, true);
Sensor rd_front2 = new ArduinoHC_SR04(rd.getName()+"front2",
ArduinoPin.PIN_48, ArduinoPin.PIN_49, true, 3, false, true);
Sensor rd_right = new ArduinoHC_SR04(rd.getName()+"right",
ArduinoPin.PIN_50, ArduinoPin.PIN_51, true, 3, false, true);
Sensor rd_right2 = new ArduinoHC_SR04(rd.getName()+"right2",
ArduinoPin.PIN_32, ArduinoPin.PIN_33, true, 3, false, true);
Sensor dq_left = new ArduinoHC_SR04(dq.getName()+"left",
ArduinoPin.PIN_30, ArduinoPin.PIN_31, true, 3, false, true);
Sensor angular_velocity = new ArduinoSERVO(av.getName(),
ArduinoPin.PIN_9, -1, 1, 45, 135, 45, true);
Sensor linear_acceleration = new ArduinoH_BRIDGE_L298N(
la.getName(), ArduinoPin.PIN_6, ArduinoPin.PIN_7,
ArduinoPin.PIN_4, ArduinoPin.PIN_2, ArduinoPin.PIN_1,
ArduinoPin.PIN_5, -1, 1, 40, 70, 15);
Sensor orientation = new ArduinoMPU6050(o.getName(), -45,
45, true);

// Paso 3: Crear los sensores agregados
ArrayList<Sensor> sensorsRD = new ArrayList<>();
sensorsRD.add(rd_front);
sensorsRD.add(rd_front2);
sensorsRD.add(rd_right);
sensorsRD.add(rd_right2);
AggregatedSensor rdSensor = new ArduinoAggregatedSensorRD(
rd.getName(), sensorsRD, 0, 3, 6, 50, true);
ArrayList<Sensor> sensorsDQ = new ArrayList<>();
sensorsDQ.add(dq_left);
AggregatedSensor dqSensor = new ArduinoAggregatedSensorDQ(
dq.getName(), sensorsDQ, 0, 2);
ArrayList<Sensor> sensorsO = new ArrayList<>();
sensorsO.add(rd_right);
sensorsO.add(rd_right2);
AggregatedSensor OSensor = new ArduinoAggregatedSensorO(
o.getName(), sensorsO, -45, 45);
ArrayList<Sensor> sensorsAV = new ArrayList<>();
sensorsAV.add(angular_velocity);
AggregatedSensor avSensor = new ArduinoAggregatedSensorAV(
av.getName(), sensorsAV, -1, 1, 18);
AggregatedSensor vSensor = new ArduinoAggregatedSensorV(
v.getName(), -1, 1, 50);

// Paso 4: Mapeo de los sensores/actuatores
ArrayList<EmbeddedVariable> embVar =
new ArrayList<>();
embVar.add(new EmbeddedVariableArduino(0, rd, rd_front));
embVar.add(new EmbeddedVariableArduino(0, rd, rd_front2));
embVar.add(new EmbeddedVariableArduino(0, rd, rd_right));
embVar.add(new EmbeddedVariableArduino(0, rd, rd_right2));
embVar.add(new EmbeddedVariableArduino(0, rd, rdSensor));
embVar.add(new EmbeddedVariableArduino(1, dq, dq_left));
embVar.add(new EmbeddedVariableArduino(1, dq, dqSensor));
embVar.add(new EmbeddedVariableArduino(2, o, orientation));
embVar.add(new EmbeddedVariableArduino(2, o, OSensor));
embVar.add(new EmbeddedVariableArduino(3, v, vSensor));
embVar.add(new EmbeddedVariableArduino(4, la,
linear_acceleration));
embVar.add(new EmbeddedVariableArduino(5, av,
angular_velocity));
embVar.add(new EmbeddedVariableArduino(5, av, avSensor));

// Paso 5: Crear el sistema embebido y el fichero .ino
EmbeddedSystem robot = new EmbeddedSystemArduino
("RobotMamdani","/dev/ttyACM0", 2000000, embVar);
robot.createRunnableEmbeddedFile("./Examples/Arduino/
RobotMamdani.ino");
```

Una vez creado el archivo *.ino*, éste se escriben en el microcontrolador de la placa Arduino mediante la IDE proporcionado por la empresa y se establece la conexión Bluetooth con el portátil. Finalmente, el sistema embebido se asocia al controlador difuso y se ejecuta el programa. Para realizar este paso se utiliza el siguiente código Java:

```
ArrayList<EmbeddedSystem> arduinos = new ArrayList<>();
arduinos.add(robot);

EmbeddedController controller = new EmbeddedControllerSystem
(arduinos, fs);
controller.run(1000);
```

Destacar que el sistema embebido es añadido a una lista porque podríamos utilizar varias placas de Arduino para utilizar los sensores/actuadores. Las pruebas fueron realizadas en la planta baja del Centro de Investigación en Tecnologías de la Información y las Comunicaciones de la Universidad de Granada (CITIC-UGR) que incluye diferentes situaciones a las que el robot se enfrenta habitualmente durante la navegación: paredes rectas de diferentes longitudes, esquinas cóncavas y convexas, etc., cubriendo así una amplia gama de contornos. La distancia máxima deseada en este caso fue de 50 cm. La Figura 2 muestra la trayectoria del robot mediante marcas rojas circulares. Podemos ver cómo el robot presenta un buen comportamiento a pesar de que las medidas de los sensores de ultrasonidos son bastante ruidosas debido a que los pasillos tienen adoquines en las paredes y las puertas son de madera. Para este entorno el robot registro una distancia media de 50,5 cm con una desviación típica de 3 cm y una velocidad media de 42,9 cm/s.

V. Conclusiones

En este trabajo presentamos un nuevo módulo Java de código abierto para la librería JFML que permite conectar fácilmente un SBRD con uno de los sistemas embebidos más utilizados en el modelo de hardware abierto, Arduino. Este módulo permite asociar las variables de entrada/salida de un SBRD con sensores/actuadores a través de una placa Arduino. Además, este módulo integra un protocolo de comunicación vía Bluetooth entre JFML y las placas Arduino

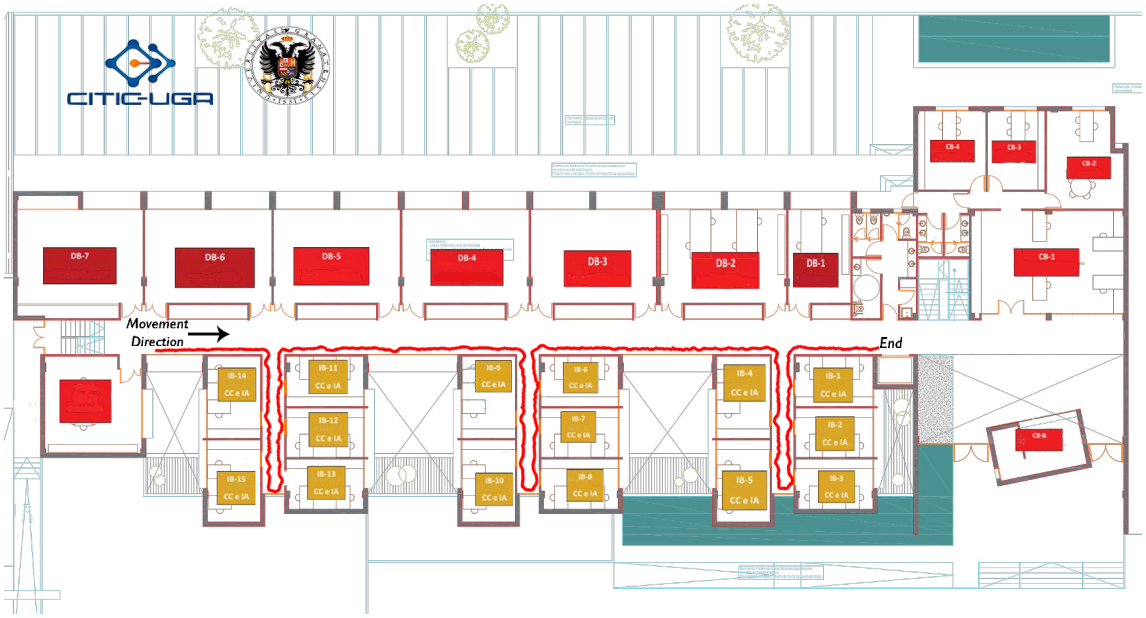

Figura 2. Entorno de prueba

que permite realizar la inferencia difusa en un ordenador externo, eliminando la limitada capacidad de cálculo que ofrecen los sistemas embebidos. Para ilustrar el potencial del nuevo módulo hemos mostrado un caso de estudio con un controlador difuso definido según IEEE Std 1855^{TM}-2016 para controlar un robot móvil para seguimiento de contornos. Se ilustra el uso del nuevo módulo en un breve programa Java donde las variables de entrada/salida del controlador se asocian con los sensores de distancia y los motores, se genera el archivo *.ino* para el microcontrolador de la placa Arduino, y se gestiona el robot realizando la inferencia difusa desde un ordenador externo manteniendo la comunicación vía Bluetooth. Actualmente se está trabajando en la incorporación de nuevos sensores y actuadores al módulo y en el desarrollo de un mecanismo de comunicación Wifi.

Referencias

[1] E. Trillas and L. Eciolaza, *Fuzzy Logic: An Introductory Course for Engineering Students*. Springer, 2015.

[2] J. Alcala-Fdez, R. Alcala, S. Gonzalez, Y. Nojima, and S. Garcia, "Evolutionary fuzzy rule-based methods for monotonic classification," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 6, pp. 1376–1390, 2017.

[3] H. Zuo, G. Zhang, W. Pedrycz, V. Behbood, and J. Lu, "Fuzzy regression transfer learning in takagi-sugeno fuzzy models," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 6, pp. 1795–1807, 2017.

[4] Y. Li, S. Sui, and S. Tong, "Adaptive fuzzy control design for stochastic nonlinear switched systems with arbitrary switchings and unmodeled dynamics," *IEEE Transactions on Cybernetics*, vol. 47, no. 2, pp. 403–414, 2017.

[5] G. Acampora, B. di Stefano, and A. Vitiello, "IEEE 1855^{TM}: The first IEEE standard sponsored by IEEE Computational Intelligence Society," *IEEE Computational Intelligence Magazine*, vol. 11, no. 4, pp. 4–7, 2016.

[6] G. Acampora, "Fuzzy Markup Language: a XML based language for enabling full interoperability in fuzzy systems design," in *On the Power of Fuzzy Markup Language*, G. Acampora, V. Loia, C.-S. Lee, and M.-H. Wang, Eds. Springer, Berlin, 2013, pp. 17–31.

[7] J. Alcalá-Fdez and J. M. Alonso, "A survey of fuzzy systems software: Taxonomy, current research trends and prospects," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 1, pp. 40–56, 2016.

[8] Arduino. (2017) Open-source electronic prototyping platform enabling users to create interactive electronic objects.

[9] J. Pearce, *Open-Source Lab: How to Build Your Own Hardware and Reduce Research Costs*. Elsevier, 2013.

[10] M. Muñoz, E. Miranda, and P. Sánchez, "A fuzzy system for estimating premium cost of option exchange using mamdani inference: Derivatives market of mexico," *International Journal of Computational Intelligence Systems*, vol. 10, no. 1, pp. 153–164, 2017.

[11] A. Fahmi, K. Ulengin, and C. Kahraman, "Analysis of brand image effect on advertising awareness using a neuro-fuzzy and a neural network prediction models," *International Journal of Computational Intelligence Systems*, vol. 10, no. 1, pp. 690–710, 2017.

[12] H. Zermane and H. Mouss, "Internet and fuzzy based control system for rotary kiln in cement manufacturing plant," *International Journal of Computational Intelligence Systems*, vol. 10, no. 1, pp. 835–850, 2017.

[13] L. Tang and J. Zhao, "Adaptive tracking control for discrete-time switched nonlinear systems with dead-zone inputs," *Fuzzy Sets and Systems*, vol. 344, pp. 51–69, 2018.

[14] B. Zhang, C. Yang, H. Zhu, P. Shi, and W. Gui, "Controllable-domain-based fuzzy rule extraction for copper removal process control," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 3, pp. 1744–1756, 2018.

[15] M. Barr, "Embedded systems glossary," *Neutrino Technical Library*, retrieved 2007-04-21.

[16] F. López-Rodríguez and F. Cuesta, "Andruino-A1: Low-cost educational mobile robot based on android and arduino," *Journal of Intelligent & Robotic Systems*, vol. 81, no. 1, pp. 63–76, 2016.

[17] A. A. Khater, M. El-Bardini, and N. M. El-Rabaie, "Embedded adaptive fuzzy controller based on reinforcement learning for dc motor with flexible shaft," *Arabian Journal for Science and Engineering*, vol. 40, no. 8, pp. 2389–2406, 2015.

[18] *International Electrotechnical Commission technical committee industrial process measurement and control. IEC 61131 - Programmable Controllers*. IEC, 2000.

[19] IEEE-SA Standards Board, "IEEE standard for fuzzy markup language," *IEEE Std 1855-2016*, pp. 1–89, 2016.

[20] K. Karvinen and T. Karvinen, *Getting Started with Sensors: Measure the World with Electronics, Arduino, and Raspberry Pi*. Maker Media, Incorporated, 2014.

[21] M. Mucientes, R. Alcalá, J. Alcalá-Fdez, and J. Casillas, "Learning weighted linguistic rules to control an autonomous robot," *International Journal of Intelligent Systems*, vol. 24, no. 3, pp. 226–251, 2009.