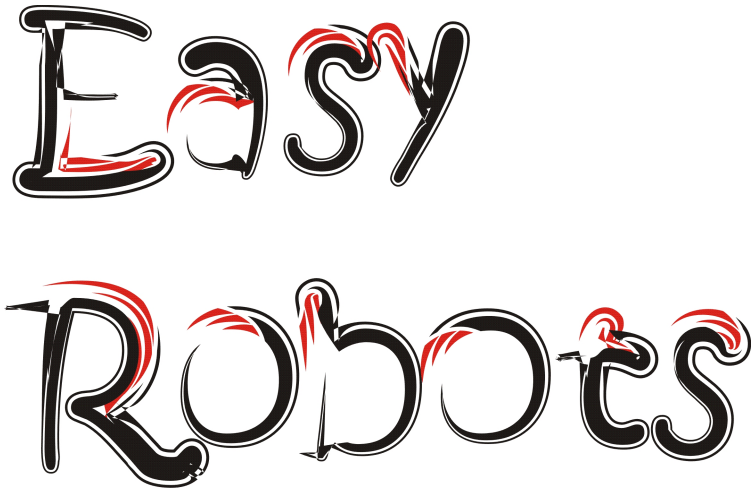
Proyecto de grado

Control y Comportamiento de Robots Omnidireccionales

22 de noviembre de 2009



Manual de Uso

Santiago Martínez, Rafael Sisto
pgomni@fing.edu.uy
http://www.fing.edu.uy/~pgomni

Tutor

Gonzalo Tejera

Cotutores

Facundo Benavides, Santiago Margni

Versión 1.2

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

Resumen

Este documento contiene las instrucciones necesarias para que un usuario pueda hacer uso de la aplicación *EasyRobots*. Para ello se incluyen el manual de instalación, configuración y ejecución de la aplicación asegurando que se pueda hacer uso de ésta en cualquier entorno de ejecución soportado así como las instrucciones para el armado de un robot omnidireccional de tres ruedas. Finalmente se incluyen las componentes lógicas posibles a utilizar en una aplicación robotica definida en *EasyRobots*.

Índice

Índice de figuras

1. Aspectos generales

1.1. Descarga

La descarga de los binarios y código fuente de la aplicación se puede realizar desde el sitio web del proyecto, dentro de la sección descargas. El link para la descarga es el siguiente:

http://www.fing.edu.uy/~pgomni/descargas.html

1.2. Descarga herramientas

1.2.1. Entorno de ejecución

Para poder ejecutar la aplicación, se requiere la máquina virtual *Java v6.0* o superior. Se recomienda la máquina virtual de *Sun Microsystems*, descargable de:

http://www.java.com/es/download/

El manual de instalación de esta herramienta para Linux y Microsoft Windows se puede encontrar en la siguiente página:

http://www.java.com/es/download/help

1.2.2. Entorno de edición

Para la edición de archivos XML (requeridos para definir aplicaciones robóticas), se recomienda utilizar el entorno de desarrollo *Eclipse*, descargable de

http://www.eclipse.org/downloads/

Este entorno de desarrollo trae incorporado un complemento llamado *Web Tools Platform (WTP)*, el cual permite, entre otras funcionalidades, definir XML gráficamente, teniendo un esquema de definición para XML (XSD).

Además de este entorno de edición, pueden utilizarse otras herramientas para edición de archivos XML, mientras éstos se validen utilizando el archivo de esquema XSD brindado con EasyRobots.

El archivo de esquema XSD de aplicaciones robóticas puede encontrarse dentro de la carpeta *static*, dentro de la carpeta de la aplicación EasyRobots.

1.3. Ejecución

Para la ejecución de la aplación en el entorno Linux, se ejecuta el archivo "*run.sh*". Puede ser ejecutado mediante doble click con el ratón sobre el ícono del archivo (en caso de utilizarse un entorno gráfico) o ejecutarlo mediante línea de comandos usando el siguiente comando:

./run.sh

Para la ejecución de la aplación en el entorno Windows, se ejecuta el archivo "*run.bat*". Puede ser ejecutado mediante doble click con el ratón sobre el ícono del archivo (en caso de utilizarse un entorno gráfico) o ejecutarlo mediante línea de comandos usando el siguiente comando:

```
run.bat
```

Por otro lado, independientemente del sistema operativo, se puede ejecutar el archivo *jar* mediante el siguiente comando (luego de ubicarse en el directorio donde se encuentra el archivo *jar*):

```
java -jar EasyRobots.jar
```

Luego de ejecutar la aplicación, se deplegará la interfaz de línea de comandos. En ésta, se disponen de cinco opciones:

1. *Cargar aplicación robótica*: Permite cargar un archivo XML conteniendo una aplicación robótica. Se puede especificar la ubicación del archivo mediante una ruta absoluta o relativa.
2. *Activar comportamiento*: Permite activar un comportamiento de los definidos en la aplicación robótica cargada. Se debe especificar su identificador.
3. *Desactivar comportamiento*: Permite desactivar un comportamiento activo. Se debe especificar su identificador.
4. *Listar comportamientos*: Lista los comportamientos presentes en la aplicación robótica, así como su estado (activado o desactivado).
5. *Salir*: Salir de la aplicación EasyRobots.

A modo de ejemplo se incluyen, en el archivo descargado del sitio web del proyecto, cuatro aplicaciones ya definidas. Éstas se encuentran en el directorio *examples*.

- La primera de ellas, *Aplication1.xml*, permite que el robot definido se dirija a un punto mediante un sensor de predicción.
- La segunda, *Aplication2.xml*, permite que el robot se dirija a la pelota utilizando el Sistema de Sensado *Doraemon*.
- La tercera, *Aplication3.xml*, permite que el robot realice una trayectoria dentada haciendo uso del Sistema de Sensado Doraemon.
- La cuarta, *Aplication4.xml*, permite que el robot realice una trayectoria cuadrada haciendo uso del algoritmo de sensor fusion: promedio ponderado.

2. Construcción de un robot omnidireccional

Para una construcción posible de un robot omnidireccional de tres ruedas, referirse al sitio web del proyecto [4], en la sección *documentos*. Allí se modeló un robot omnidireccional utilizando la aplicacion de modelado *Lego Digital Designer* [1] y se listan los pasos a seguir para su construcción. El robot modelado es el de la figura 1, con la salvedad que las ruedas mostradas no corresponden a ruedas especiales utilizadas ya que no son modelables en la aplicación utilizada.

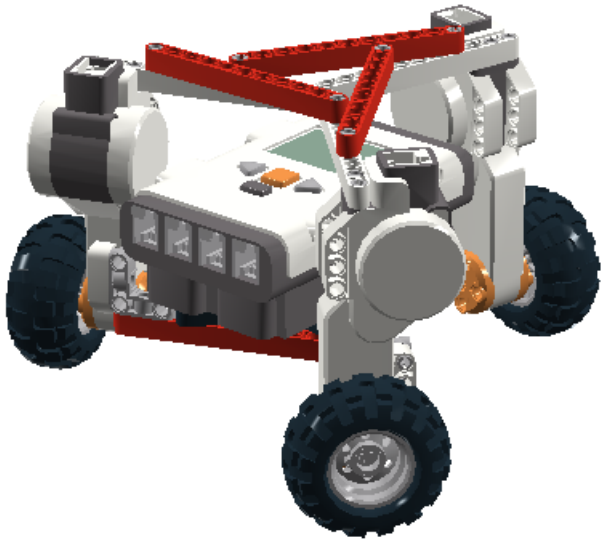


Figura 1: Modelo en Lego Digital Designer [1] de un robot omnidireccional de tres ruedas.

3. Creación de aplicaciones robóticas

En esta sección se explican los atributos necesarios para definir aplicaciones robóticas en el sistema *EasyRobots*. Se explicarán al detalle las componentes que conforman la aplicación: Controladores de Robots Físicos, Entidades, Sensores lógicos y Comportamientos. Las componentes que aquí se detallan son las incluidas en la última versión de la aplicación. En caso de que se desee hacer uso de nuevas componentes, se debe extender la aplicación, tal como se detalla en la Descripción de la Arquitectura del Sistema EasyRobots [3].

3.1. Creación de aplicaciones robóticas

El archivo XML que defina una aplicación, debe contener el siguiente código en su comienzo:

```
<?xml
   version="1.0"
   encoding="UTF-8"?>
   <application
      xmlns="http://www.example.org/RobotApp"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.example.org/RobotApp RobotApp.xsd ">
```

Este código define la ubicación del esquema, el cual es utilizado por la aplicación para validar el archivo. Este código es agregado por defecto si se genera el archivo XML a partir del esquema haciendo uso del *plug-in* de *Eclipse*.

Luego de definir las componentes en el archivo, se debe incluir el siguiente código a fin de indicar el final de la definición de la aplicación robótica.

```
</application>
```

Tras la indicación de comienzo de aplicación, se comienza la definición de las componentes de la aplicación robótica.

3.2. Creación de Controladores de Robots Físicos

A fin de establecer la comunicación de la aplicación con los Sistemas de Actuado, se deben crear controladores de robots físicos. Para cada Sistema de Actuado, se debe crear un controlador.

En la versión de la aplicación desarrollada, el único tipo incluído es *NXTController*, que indica que es un controlador de robots de tipo Lego NXT [2]. La definición de un controlador de Lego NXT, incluye los siguientes atributos:

- Tipo (atributo *type*): Indica el tipo de controlador que se definirá. En la definición del archivo XML, el atributo debe poseer el sufijo "*xsi:*", que indica que el tipo determina los atributos.
- Identificador (atributo *id*): Indica el identificador del controlador que se define, para luego ser utilizado en la aplicación robótica.
- Dirección de red (atributo *ip*): Indica la dirección de red de la computadora donde se encuentra el Sistema de Actuado.
- Puerto (atributo *port*): Indica el puerto de red que posee el Sistema de Actuado en la computadora donde se ejecuta.

El código XML que define un controlador es el siguiente:

```
   <robotcontroller
      xsi:type="Tipo"
      id="Identificador"
      port="Número de puerto"
      ip="Dirección de red" />
```

El siguiente es un ejemplo del código XML que define un controlador de Lego NXT:

```
<robotcontroller
   xsi:type="NXTController"
   id="nxtController"
   port="1234"
   ip="192.168.32.29" />
```

3.3. Creación de Entidades

Para modelar las entidades de la aplicación robótica, se poseen varios tipos de entidades lógicas. Los tipos de entidad están determinados por el atributo *type*.

Para las entidades, el atributo *type* puede variar entre *GeneralEntity*, para indicar que se trata de una entidad general, *OmnidirectionalRobot* para modelar un robot omnidireccional o *BiWheeledRobot* para un robot de dos ruedas. Para cualquier tipo de entidad, se poseen los siguientes atributos comunes:

- Tipo (atributo *type*): Tipo de entidad.
- Identificador (atributo *id*): Este atributo indica el identificador de la entidad.
- Identificador del sensor lógico (atributo *positionId*): Este atributo indica el identificador del sensor lógico que se utilizará para conocer la posición y estado de la entidad.

3.3.1. Entidad General

En caso de que se trate de una entidad general (por ejemplo: una pelota, un robot oponente o un obstáculo), los atributos que la definen son:

- Tipo (atributo *type*): Indica el tipo de la entidad. En este caso, es *GeneralEntity*.

El código XML que define este tipo de entidad es el siguiente:

```
<entity
   xsi:type="GeneralEntity"
   id="Identificador"
   positionId="Identificador del sensor lógico" />
```

El siguiente es un código XML de ejemplo que define una entidad general:

```
<entity
   xsi:type="GeneralEntity"
   id="entityId1"
   positionId="sensorId1" />
```

3.3.2. Entidad Robot Omnidireccional

En caso de un robot omnidireccional, los atributos que lo definen son:

- Tipo (atributo *type*): Tipo de entidad. En este caso, es *OmnidireccionalRobot*.
- Velocidad angular máxima (atributo *maxwheelrotation*): Velocidad angular máxima de las ruedas (en radianes por segundo). Atributo opcional, en caso de que se quiera restringir la velocidad de los motores.
- Velocidad máxima (elemento múltiple *maxspeed*): Velocidad máxima (en metros por segundo) del robot en el plano $\left(\dot{x}, \dot{y}, \dot{\theta}\right)$. Atributo opcional.

- Rueda (elemento compuesto *wheel*): Este atributo contiene los atributos necesarios para definir una rueda, y así la estructura del robot (ver figura 2). Estos son:
  - Radio (atributo *radious*): Radio (en metros) de la rueda, considerando la distancia de su eje al punto de apoyo a la superficie.
  - Distancia del centro (atributo *d*): Distancia del centro del robot a la rueda.
  - Ángulo rueda-rodillo (atributo *alpha*): Ángulo (en radianes) entre el eje de la rueda y sus rodillos.
  - Ángulo eje *OX*-rueda (atributo *beta*): Ángulo (en radianes) entre el eje *OX* en coordenadas cartesianas y la rueda.
  - Ángulo eje *OX*-eje rueda (atributo *gamma*): Ángulo (en radianes) entre el eje *OX* en coordenadas cartesianas y el eje de la rueda.
- Tipo de robot físico (elemento compuesto *physicalRobot*): Este atributo contiene los atributos necesarios para identificar el tipo de robot físico que es modelado por la entidad. Éstos son:
  - Tipo (atributo *type*): Tipo de robot físico. En la versión de la aplicación entregada, puede ser *NXTRobot*.
  - Dirección física del robot (atributo *physicalAddress*): Dirección física del robot (Bluetooth), de forma de identificarlo.
  - Identificador del controlador (atributo *physicalControllerId*): Identificador del controlador (ver sección 3.2) que se utilizará para enviar comandos al robot.

La figura 2 permite comprender los atributos de definición de una rueda en mayor detalle:

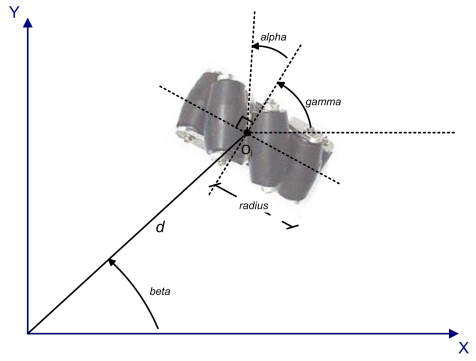


Figura 2: Parámetros de la definición de una rueda

El código XML que define este tipo de entidad (robot omnidireccional de tres ruedas) es el siguiente:

```
<entity
  xsi:type="OmnidirectionalRobot"
   id="Identificador"
   maxwheelrotation="Velocidad máxima de rotación"
   positionId="Identificador del sensor lógico" >
   <physicalRobot
      xsi:type="NXTRobot"
      physicalAddress="Dirección física"
      physicalControllerId="Identificador del controlador">
   </physicalRobot>
   <maxspeed>0.0</maxspeed>
   <maxspeed>0.0</maxspeed>
   <maxspeed>0.0</maxspeed>
   <wheel
      radius="Radio"
```

```
        gamma="Ángulo entre eje OX y eje de la rueda"
        beta="Ángulo entre eje OX y eje la rueda"
        alpha="Ángulo entre la rueda y sus rodillos"
        d="Distancia de la rueda al centro" />
    <wheel
        radius="Radio"
        gamma="Ángulo entre eje OX y eje de la rueda"
        beta="Ángulo entre eje OX y eje la rueda"
        alpha="Ángulo entre la rueda y sus rodillos"
        d="Distancia de la rueda al centro" />
    <wheel
        radius="Radio"
        gamma="Ángulo entre eje OX y eje de la rueda"
        beta="Ángulo entre eje OX y eje la rueda"
        alpha="Ángulo entre la rueda y sus rodillos"
        d="Distancia de la rueda al centro" />
</entity>
```

El siguiente es un código XML de ejemplo que define una robot omnidireccional de tres ruedas:

```
<entity
    xsi:type="OmnidirectionalRobot"
    id="robotId1"
    maxwheelrotation="0.4"
    positionId="sensorId1" >
    <physicalRobot
        xsi:type="NXTRobot"
        physicalAddress="00:16:53:01:4E:3B"
        physicalControllerId="nxtController">
    </physicalRobot>
    <maxspeed>1.1</maxspeed>
    <maxspeed>0.5</maxspeed>
    <maxspeed>1.0</maxspeed>
    <wheel
        radius="0.032"
        gamma="0.0"
        beta="0.0"
        alpha="1.57079633"
        d="0.095" />
    <wheel
        radius="0.032"
        gamma="2.0943951"
        beta="2.0943951"
        alpha="1.57079633"
        d="0.095" />
    <wheel
        radius="0.032"
        gamma="4.1887902"
        beta="4.1887902"
        alpha="1.57079633"
        d="0.095" />
</entity>
```

3.3.3. Entidad Robot de dos Ruedas

En caso de un robot de dos ruedas, los atributos que lo definen son:

- Tipo (atributo *type*): Tipo de entidad. En este caso, es *BiWheeledRobot*.
- Velocidad angular máxima (atributo *maxwheelrotation*): Velocidad angular máxima de las ruedas (en radianes por segundo). Atributo opcional, en caso de que se quiera restringir la velocidad de los motores.
- Distancia de las ruedas (atributo robotRadius): Distancia (en metros) de cada rueda al centro del robot.
- Radio de las ruedas (atributo robotWheelRadius): Radio (en metros) de cada rueda.

El código XML que define este tipo de entidad es el siguiente:

```
<entity
    id="Identificador"
    xsi:type="BiWheeledRobot"
    maxwheelrotation="Velocidad máxima de rotación"
    positionId="Identificador del sensor lógico"
    robotRadius="Radio del robot"
    robotWheelRadius="Radio de ruedas" >
</entity>
```

El siguiente es un código XML de ejemplo que define una robot omnidireccional de tres ruedas:

```
<entity
    id="idRobot1"
    xsi:type="BiWheeledRobot"
    maxwheelrotation="0.4"
    positionId="sensorId1"
    robotRadius="0.1"
    robotWheelRadius="0.032" >
</entity>
```

3.4. Creación de Sensores Lógicos

Para modelar sensores físicos y su comunicación con la aplicación, se cuenta con diversos tipos de sensores lógicos. Los tipos de sensores están determinados por el atributo *type*. De acuerdo a este atributo, se determinan los otros atributos.

El atributo type puede poseer el valor *DoraemonClient* que indica que el sensor es utilizado para recibir mediciones del Sistema de Sensado *Doraemon*, *PredictionSensor* que indica que el sensor es utilizado como sensor predictor de posiciones y *AverageSensorFusion* que indica que el sensor realiza la fusión por medio de promedio ponderado de otros sensores. Los diversos tipos de sensores poseen un conjunto de atributos comunes, a saber:

- Tipo (atributo *type*): Tipo de sensor lógico.
- Confianza (atributo *confidence*): Confianza en las mediciones del sensor, utilizada en la fusión de sensores. Su valor varía en el rango de 0 a 1, donde 0 es la confianza más baja y 1 es la más alta.
- Período (atributo period): Período, en milisegundos, entre consultas al sensor físico.
- Identificador (atributo *id*): Identificador del sensor lógico.

El siguiente código XML define este tipo de sensor:

```
<sensor
    confidence="Confianza"
    period="Período"
    id="Identificador"
    xsi:type="Tipo de sensor" />
```

El siguiente es un código XML de ejemplo que define un sensor de este tipo:

```
<sensor
    confidence="0.5"
    period="20"
    id="sensorId1"
    xsi:type="Tipo abstracto" />
```

3.4.1. Sensor Predictor

En el caso que el sensor sea de tipo predictor, es decir que el atributo *type* posee el valor *PredictionSensor*, los atributos que lo definen son:

- Tipo (atributo type): En este caso, *PredictionSensor*.
- Identificador de la entidad controlada (atributo *managedEntityId*): Identificador de la entidad sensada por el sensor lógico.
- Identificador del sensor del cual se obtienen posiciones (atributo *referenceSensorId*): Identificador del sensor del cual se obtiene la posición de la entidad controlada, para luego predecir la nueva posición (Puede ser el mismo que el identificador propio).

El siguiente código XML define este tipo de sensor:

```
<sensor
    confidence="Confianza"
    period="Período"
    id="Identificador"
    xsi:type="PredictionSensor"
    managedEntityId="Entidad controlada"
    referenceSensorId="Sensor referenciado" />
```

El siguiente es un código XML de ejemplo que define un sensor de este tipo:

```
<sensor
    confidence="0.5"
    period="0.01"
    id="sensorId1"
    xsi:type="PredictionSensor"
    managedEntityId="entityId1"
    referenceSensorId="sensorId2" />
```

3.4.2. Fusión de Sensores (Sensor Fusion)

En la versión de la aplicación entregada, se posee la posibilidad de definir sensores de fusión de datos mediante promedio ponderado de sus mediciones (*AverageSensorFusion*) o mediante la selección de un sensado de un conjunto posible (*FashionSensorFusion*). Los atributos comunes a los algoritmos de fusión de sensores son:

- Tipo (atributo *type*): En este caso, *AverageSensorFusion o FashionSensorFusion*.

- Identificador de la entidad controlada (atributo *managedEntityId*): Identificador de la entidad sensada por el sensor lógico.
- Identificador de sensor fusionado (elemento *fusionedSensorId*): Identificadores de los sensores fusionados. Por cada sensor que se fusione, se debe incluir una instancia de este atributo.

AverageSensorFusion En caso de utilizar la fusión de datos mediante promedio ponderado, el atributo type debe poseer el valor *AverageSensorFusion*. Este tipo de fusión no posee atributos adicionales a los de un algoritmo de fusión general.

El siguiente código XML define este tipo de sensor:

```
<sensor
    confidence="Confianza"
    period="Período"
    id="Identificador"
    xsi:type="AverageSensorFusion"
    managedEntityId="Entidad controlada" >
    <fusionedSensorId>Identificador de sensor fusionado</fusionedSensorId>
</sensor>
```

El siguiente es un código XML de ejemplo que define un sensor de este tipo. En el ejemplo, se fusionan dos sensores identificados por *sensorId1* y *sensorId2*.

```
<sensor
    confidence="1"
    period="0.3"
    id="idSensorFusion"
    xsi:type="AverageSensorFusion"
    managedEntityId="idEntidad1" >
    <fusionedSensorId>sensorId1</fusionedSensorId>
    <fusionedSensorId>sensorId2</fusionedSensorId>
</sensor>
```

FashionSensorFusion En caso de utilizar este tipo de fusión de datos, el atributo type debe poseer el valor *FashionSensorFusion*. Este tipo posee un único atributo adicional al de un algoritmo de fusión general:

- Ventana de validez (atributo *checkThreshold*): Ventana temporal (en milisegundos) utilizada para considerar si un sensor actualizó su valor sensado.

El siguiente código XML define este tipo de sensor:

```
<sensor
    confidence="Confianza"
    period="Período"
    id="Identificador"
    checkThreshold="Ventana de validez"
    xsi:type="FashionSensorFusion"
    managedEntityId="Entidad controlada" >
    <fusionedSensorId>Identificador de sensor fusionado</fusionedSensorId>
</sensor>
```

El siguiente es un código XML de ejemplo que define un sensor de este tipo. En el ejemplo, se fusionan dos sensores identificados por *sensorId1* y *sensorId2*.

```
<sensor
    confidence="1"
    period="0.3"
    id="idSensorFusion"
    checkThreshold="50"
    xsi:type="FashionSensorFusion"
    managedEntityId="idEntidad1" >
    <fusionedSensorId>sensorId1</fusionedSensorId>
    <fusionedSensorId>sensorId2</fusionedSensorId>
</sensor>
```

El orden de consulta por la disponibilidad de cada sensor se determina por el orden de los sensores utilizados, definidos haciendo uso del elemento *fusionedSensorId*. En el ejemplo anterior, se consulta por la disponibilidad del sensor identificado por sensorId1. Si éste se encuentra disponible, se utiliza su sensado, en caso contrario se continúa la consulta con la lista de sensores restantes, en este caso sensorId2. Si ningún sensor se encuentra disponible, se detienen los robots controlados.

3.4.3. Sensor Doraemon

Este tipo de sensor es utilizado para modelar la comunicación con el Sistema de Sensado *Doraemon*. Sus atributos son:

- Tipo (atributo *type*): En este caso, *DoraemonClient*.
- Largo máximo de mensaje (atributo *maxMessageLength*): Largo máximo, en *bytes*, de cada mensaje que se recibirá del Sistema de Sensado.
- Puerto (atributo *portServer*): Puerto de broadcast a través del cual se recibe el mensaje del Sistema de Sensado.
- Correspondencia objeto-identificador (elemento compuesto *idDoraemonToEntity*): Correspondencia del objeto definido en el servidor Doraemon con el identificador interno de dicha entidad en el Sistema de Sensado. Para cada entidad controlada, se debe crear una instancia de este atributo y realizar la correspondencia. Posee los siguientes atributos:
  - Identificador interno en Doraemon (atributo *string1*): Identificador de la entidad en el *Doraemon*.
  - Identificador en EasyRobots (atributo *string2*): Identificador de la entidad en EasyRobots.

El siguiente código XML define este tipo de sensor:

```
<sensor
    confidence="Confianza"
    period="Período"
    id="Identificador"
    xsi:type="DoraemonClient"
    maxMessageLength="Largo máximo del mensaje"
    portServer="Puerto" >
    <idDoraemonToEntity
        string1="Identificador en Doraemon"
        string2="Identificador en EasyRobots" />
</sensor>
```

El siguiente código XML es un ejemplo del uso de este tipo de sensor. En el ejemplo, el Sistema de Sensado *Doraemon* registra posiciones de dos entidades. Una de éstas, *idEntity1*, se identifica como *OBJ1* en el Doraemon, mientras la otra, *idEntity2*, se identifica en el Doraemon como *OBJ2*.

```
<sensor
    confidence="0.8"
    period="0.05"
    id="sensorId"
    xsi:type="DoraemonClient"
    maxMessageLength="512"
    portServer="1234" >
    <idDoraemonToEntity
        string1="OBJ1"
        string2="idEntity1" />
    <idDoraemonToEntity
        string1="OBJ2"
        string2="idEntity2" />
</sensor>
```

3.5. Creación de Comportamientos

En la versión de la aplicación entregada, se cuenta con tres tipos diferentes de comportamientos, diferenciados por el atributo *type*. Si el valor de dicho atributo es *WaypointsStrategy*, se trata de un comportamiento de seguimientos de puntos definidos en el campo . Si su valor es *PotentialFieldsStrategy*, se trata de un comportamiento de Campos Potenciales. Finalmente, si su valor es *JavaStrategy*, se indica que se utilizará un comportamiento definido en un archivo *Java*.

Indiferentemente al tipo de comportamiento que se defina, los comportamientos tienen los siguientes atributos en común:

- Tipo (atributo *type*): Tipo del comportamiento.
- Identificador (atributo id): Identificador del comportamiento que se define.
- Robots controlados (elemento *controlledRobot*): Identificador del robot controlado por el comportamiento que se define. Se pueden definir multiples instancias de este elemento para definir un conjunto de robots.

El siguiente código XML define un comportamiento genérico:

```
<strategy
    id="Identificador"
    xsi:type="Tipo genérico">
    <controlledRobot>controlledRobot1</controlledRobot>
    <controlledRobot>controlledRobot2</controlledRobot>
</strategy>
```

3.5.1. Comportamiento Waypoints

El comportamiento de tipo Waypoints modela aquell comportamiento que determina un circuito, definido por una serie de puntos, a ser recorrido por un robot de forma ordenada de acuerdo a su orden de definición. Los atributos de este comportamiento son:

- Rango de error aceptable (atributo *epsilon*): Distancia (en metros) máxima del robot a un punto que es considerada para determinar si el punto fue alcanzado.
- Vueltas (atributo *laps*): Cantidad de veces que el robot recorrerá el circuito.
- Ciclico (atributo *loop*): Atributo *booleano (true, false)*. Determina si el circuito es cíclico. En caso de que su valor sea *true*, el atributo *laps* no es considerado.
- Velocidad (atributo *speed*): Velocidad (en metros por segundo) con la que el robot recorrerá el circuito.
- Punto (elemento complejo *waypoints*): Punto que integra el circuito. Su definición puede ser variable:

- Punto absoluto: Su definición (en coordenadas cartesianas de dos dimensiones) se realiza a través del uso de dos atributos de posición llamados *position*. El primero define la coordenada en el eje OX y el segundo en el eje OY.
- Punto relativo: Su definición se realiza respecto a la posición de una entidad. Para ello, se especifica el identificador de la entidad mediante el atributo *id*. De esta forma, se utilizará la posición de la entidad en el momento de evaluar el comportamiento.

El siguiente código XML define este tipo de comportamiento (observar que se definen dos *waypoints*, cada uno con un tipo de difinición diferente):

```
<strategy
   id="Identificador"
   xsi:type="WaypointsStrategy"
   epsilon="Distancia máxima de error"
   laps="Vueltas"
   loop="Ciclico"
   speed="Velocidad">
   <controlledRobot>Robot controlado</controlledRobot>
   <waypoint>
      <position>Coordenada (en metros) en OX</position>
      <position>Coordenada (en metros) en OY</position>
   </waypoint>
   <waypoint>
      <id>Identificador de entidad</id>
   </waypoint>
</strategy>
```

El siguiente es un código XML de ejemplo que define un comportamiento de este tipo:

```
<strategy
   id="strategyId"
   xsi:type="WaypointsStrategy"
   epsilon="0.1"
   laps="3"
   loop="false"
   speed="0.3">
   <controlledRobot>idRobot1</controlledRobot>
   <controlledRobot>idRobot2</controlledRobot>
   <waypoint>
      <position>0.3</position>
      <position>0.7</position>
   </waypoint>
   <waypoint>
      <id>idRobot3</id>
   </waypoint>
   <waypoint>
      <id>idPelota</id>
   </waypoint>
</strategy>
```

3.5.2. Comportamiento PotentialFields

El *comportamiento* de tipo PotentialFields modela aquell Estrategia que se define a través del uso de campos potenciales. Los atributos de este tipo son:

- Campo (elemento compuesto *field*): Campo potencial perteneciente al comportamiento. Sus atributos son:

  - Tipo (atributo *type*): Determina el tipo de campo. Sus valores pueden ser *FieldCircle*, para indicar un campo generado por un punto, o *FieldSemiplane*, para determinar un campo generado por una semi-recta, o *FieldUniform*, para determinar un campo uniforme.

De acuero al tipo del atributo *field*, la definición varía:

- Para definir un *field* de tipo *FieldCircle*, se requieren los siguientes atributos:

  - Alcance máximo (atributo *maxWidth*): Determina el alcance máximo (en metros) del campo potencial generado.
  - Alcance mínimo (atributo *minWidth*): Determina el alcance mínimo (en metros) del campo potencial generado.
  - Valor del campo en alcance máximo (atributo *powEnd*): Determina el valor del campo potencial en los puntos que se encuentran a una distancia del punto generador equivalente al alcance máximo del campo.
  - Valor del campo en alcance mínimo (atributo *powStart*): Determina el valor del campo potencial en los puntos que se encuentran a una distancia del punto generador equivalente al alcance mínimo del campo.
  - Ubicación del punto generador (atributo *dot*): Ubicación del campo. Su definición se realiza de forma similar a la de un waypoint para el comportamiento Waypoints (ver subsección 3.5.1), por lo que su posición puede ser estática o dependiente de la posición de una entidad.

  Es de remarcar que el valor del campo en un punto que se ubique a una distancia menor a *maxWidth* y mayor a *minWidth*, se determina interpolando linealmente los valores *powEnd* y *powStart* de acuerdo a su posición.

  Si se desea que el campo sea atractor al punto generador, los valores de *powStart* y *powEnd* deben ser negativos y lo opuesto para el caso de los detractores (obstáculos a evitar).

  La figura 3 representa los atributos de este tipo de campo:

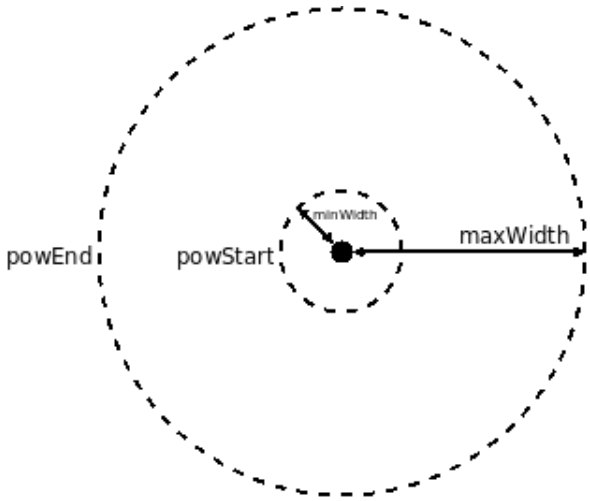


Figura 3: Atributos que definen el campo potencial circular.

- Para definir un *field* de tipo *FieldSemiPlane*, se requieren los siguientes atributos:

  - Ángulo (atributo *angle*): Determina el ángulo (en radianes) del campo generado respecto de la semi-recta generadora.
  - Alcance máximo (atributo *maxWidth*): Determina el alcance máximo (en metros) del campo potencial generado.
  - Alcance mínimo (atributo *minWidth*): Determina el alcance mínimo (en metros) del campo potencial generado.

- Valor del campo en alcance máximo (atributo *powEnd*): Determina el valor del campo potencial en los puntos que se encuentran a una distancia de la recta equivalente al alcance máximo del campo.
- Valor del campo en alcance mínimo (atributo *powStart*): Determina el valor del campo potencial en los puntos que se encuentran a una distancia de la recta equivalente al alcance mínimo del campo.
- Ubicación de la semi-recta generadora (atributo múltiple y complejo *dot*): Determina la ubicación de la semi-recta. Su ubicación es determinada por los dos puntos (P_1 y P_2) , cada uno definido mediante un *dot*. El semi-plano donde el campo tiene un valor mayor a 0, es el definido por los puntos (P_X) de la superficie para los cuales el seno del ángulo definido por $P_1P_2\hat{,}P_1P_x$es positivo.

De forma similar que para el campo de tipo FieldCircle, el valor del campo en un punto a una distancia de la semi-recta que se encuentra entre los valores *maxwidth* y *minwidth* se obtiene a partir de la interpolación del valor del campo en dichas distancias. Nuevamente, si se desea que el campo sea atractor hacia la semi-recta, los valores de *powStart* y *powEnd* deben ser negativos y viceversa.

La figura representa los atributos de este tipo de campo:

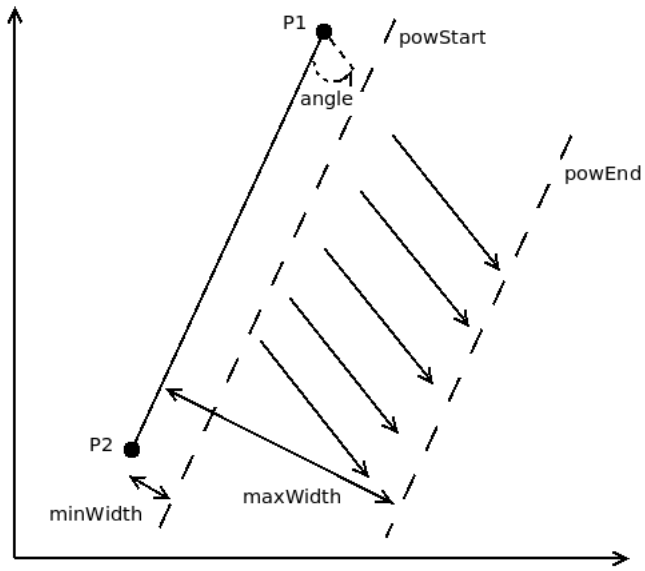


Figura 4: Atributos que definen el campo potencial generado por una semi-recta.

- Para definir un *field* de tipo *FieldUniform*, se requieren los siguientes atributos:
  - Valor en el eje OX (atributo vX): Valor del campo en el sentido del eje OX.
  - Valor en el eje OY (atributo vY): Valor del campo en el sentido del eje OY.

El siguiente código XML define este tipo de comportamiento (notar que se definen tres campos potenciales, uno de cada tipo de los explicados):

```
<strategy
   id="Identificador estrategia"
   xsi:type="PotentialFieldsStrategy">
   <controlledRobot>Identificador robot controlado</controlledRobot>
   <field
      xsi:type="FieldCircle"
      maxWidth="Alcance máximo"
      minWidth="Alcance mínimo"
      powEnd="Valor del campo en alcance máximo"
      powStart="Valor del campo en alcance mínimo">
```

```
        <dot>
            <position>Posición en el eje OX</position>
            <position>Posición en el eje OY</position>
        </dot>
    </field>
    <field
        xsi:type="FieldSemiPlane"
        angle="Ángulo"
        maxWidth="Alcance máximo"
        minWidth="Alcance mínimo"
        powEnd="Valor del campo en alcance máximo"
        powStart="Valor del campo en alcance mínimo">
        <dot>
            <position>Inicio semi-recta en el eje OX</position>
            <position>Inicio semi-recta en el eje OY</position>
        </dot>
        <dot>
            <id>Identificador de entidad. Su posición determina el fin de la semi-recta.</position>
        </dot>
    </field>
    <field
        xsi:type="FieldUniform"
        vX="Valor en el sentido del eje OX"
        vY="Valor en el sentido del eje OY">
    </field>
</strategy>
```

El siguiente es un código XML de ejemplo que define un comportamiento de este tipo:

```
<strategy
    id="strategyId"
    xsi:type="PotentialFieldsStrategy">
    <controlledRobot>idRobot1</controlledRobot>
    <controlledRobot>idRobot2</controlledRobot>
    <field
        xsi:type="FieldCircle"
        maxWidth="0.8"
        minWidth="0.0"
        powEnd="15"
        powStart="10">
        <dot>
            <position>0.5</position>
            <position>0.6</position>
        </dot>
    </field>
    <field
        xsi:type="FieldSemiPlane"
        angle="1.0"
        maxWidth="0.7"
        minWidth="0.1"
        powEnd="5"
        powStart="10">
        <dot>
            <position>0.3</position>
```

```
            <position>0.7</position>
        </dot>
        <dot>
            <id>idPelota</position>
        </dot>
    </field>
    <field
        xsi:type="FieldUniform"
        vX="3"
        vY="1">
    </field>
</strategy>
```

3.5.3. ComportamientoJava

El comportamiento Java permite modelar cualquier comportamiento mediante su definición en un archivo *Java*. Sus atributos son:

- Ruta del archivo (atributo *filePath*): Indica la ruta a la clase *Java* (archivo conteniendo el código fuente). Puede ser una ruta absoluta o relativa.

- Entidades referenciadas (elemento múltiple referencedEntity): Indica los identificadores de las entidades utilizadas en el cálculo del comportamiento. Para cada entidad referenciada se debe definir una instancia de este atributo.

El siguiente código XML define este tipo de comportamiento:

```
<strategy
    id="Identificador"
    xsi:type="JavaStrategy"
    filePath="Ruta al archivo de definición">
    <controlledRobot>Robot controlado</controlledRobot>
    <referencedEntity>Entidad referenciada</referencedEntity>
</strategy>
```

El siguiente es un código XML de ejemplo que define un comportamiento de este tipo:

```
<strategy
    id="javaId"
    xsi:type="JavaStrategy"
    filePath="/home/user/Comportamiento.java">
    <controlledRobot>idRobot1</controlledRobot>
    <controlledRobot>idRobot2</controlledRobot>
    <referencedEntity>idPelota</referencedEntity>
    <referencedEntity>idContrincante</referencedEntity>
</strategy>
```

La estructura de la clase a definir debe ser la siguiente:

```
public class Estrategia extends com.easyRobots.engine.strategy.AbstractStrategy{
    //Código adicional creado por el usuario

    @Override
    public java.util.Map<String, double[]> getSpeeds(java.util.Map<String, dou-
ble[]> positions, java.util.Map<String, double[]> speeds) {
        // Código del comportamiento
        return resultado_del_comportamiento;
    }
}
```

Como se puede observar, el usuario debe impelementar la función *getSpeeds* a fin de retornar las velocidades a asignar a cada robot a partir de las posiciones de las entidades en la superficie. La variable a retornar debe ser del tipo *java.util.Map<String, double[]>*, conteniendo para cada entidad controlada (*String* identificador), la velocidad deseada (*double[]* con las velocidades $\left(\dot{x}, \dot{y}, \dot{\theta}\right)$).

Referencias

[1] The Lego Group. Lego digital designer. Retrieved July 2009 http://ldd.lego.com/.

[2] Mindstorms. Mindstorms nxt kit. Retrieved April 2008 http://mindstorms.lego.com/Products/Default.aspx.

[3] Rafael Sisto and Santiago Martínez. Descripción de arquitectura de software. Technical report, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2009.

[4] Rafael Sisto and Santiago Martínez. Página principal del proyecto de grado: Control y comportamiento de robots omnidireccionales. Retrieved on August 2009 http://www.fing.edu.uy/ pgomni, August 2009.