ERBPI DOC

Easy Robot Behaviour Programming Interface
Documentación

Desarrollo de una interfaz
de programación para talleres
de Robótica Educativa

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

Javier Caccavelli jcaccav@dc.uba.ar
Sol Pedre spedre@dc.uba.ar
Pablo de Cristóforis pdecris@dc.uba.ar
Andrea Katz akatz@dc.uba.ar
Diego Bendersky dbenders@dc.uba.ar

Agradecimientos

A QUIEN LE AGRADECEMOS??

Contents

Chapter 1

Introducción

Realizamos un survey de interfaces gráficas de programación.

Estos son todos proyectos para niños, la mayoría implementados para la OLPC (One Laptop Per Child Project):

- StarLogo TNG
  http://education.mit.edu/drupal/starlogo-tng
- EToys (Smalltalk/Squeak)
  http://wiki.laptop.org/go/Etoys
  http://www.squeakland.org/download/
- Scratch (Squeak)
  http://scratch.mit.edu/

También hay un framework de Microsoft:

- Microsoft Robotics Developer Studio (RDS)
  http://msdn.microsoft.com/en-us/library/cc998476.aspx
- Microsoft Visual Programming Language (VPL)
  http://msdn.microsoft.com/en-us/library/bb483088.aspx
- Microsoft Visual Simulation Environment (VSE)
  http://msdn.microsoft.com/en-us/library/bb483076.aspx

ARREGLAR ESTO, COMPLETAR CON LO DEL EUROBOT2011 QUE ESTÁ MUY BIEN...

Chapter 2

ERBPI: Easy Robot Behaviour Programming Interface

La idea general del software es que permita, a través de una interfaz gráfica y sencilla, programar los robots para realizar distintas experiencias de vehículos de Braitenberg y comportamiento basado en subsumisión[1].

Para eso, nos basamos en el survey que realizamos, principalmente los programas *StarLogo* y *Scratch*, que resultaron los mejorcitos en cuanto a la interfaz gráfica de programación y la idea de la interfaz gráfica de proveer menus y submenúes con los objetos predefinidos para ir agregando...

La idea es poder combinar *Braitenberg* y *Subsumisión*, de manera que cada estado de la maquina de estados de subsimisión sea un braitenberg. Entonces, en principio, se puede hacer sólo Braitenberg. Luego, se puede hacer Subsumisión "insertando" en cada estado un braitenberg definido anteriormente.

Algo así como la Figura 2.1:

ARREGLAR ESTO, COMPLETAR CON LO DEL EUROBOT2011 QUE ESTÁ MUY BIEN...

[1] *Subsumption Architecture*, Behavior-Based Robotics, R. C. Arkin.

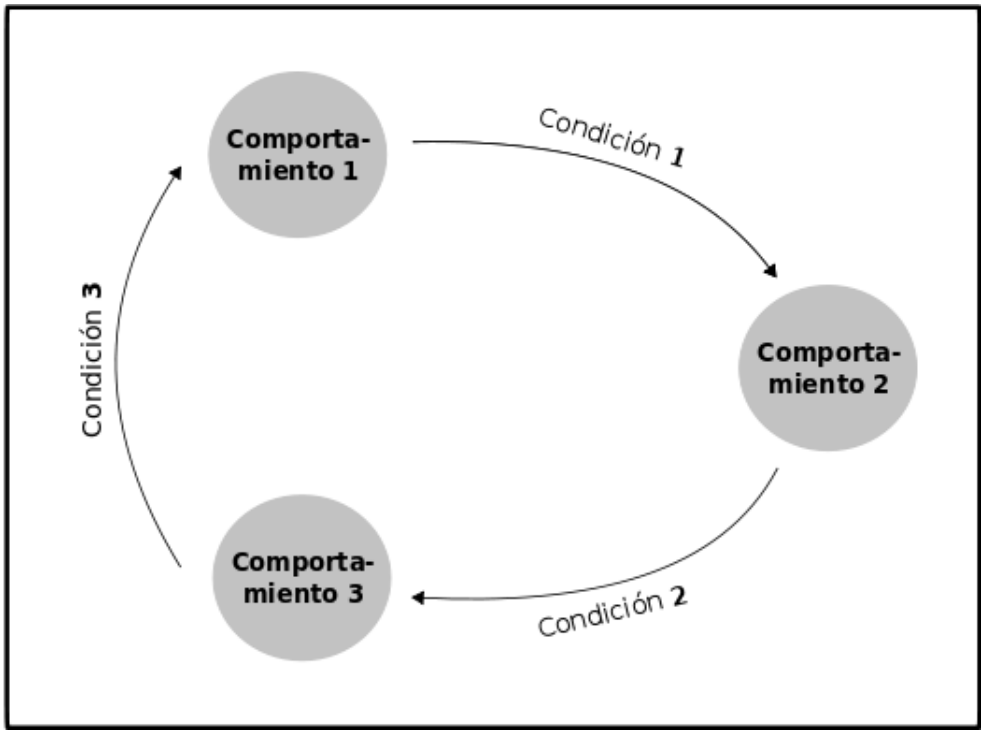

Figure 2.1: FORMA GENERAL DISEñO SOFTWARE.

Chapter 3

Diseño

El software estaría compuesto por tres módulos independientes. Un módulo *Core*, un módulo *RAL* y un módulo *GUI*. Esto nos permite realizar el desarrollo de cada módulo completamente por separado.

Algo así como la Figura 3.1:

ARREGLAR ESTO, COMPLETAR CON LO DEL EUROBOT2011 QUE ESTÁ MUY BIEN...

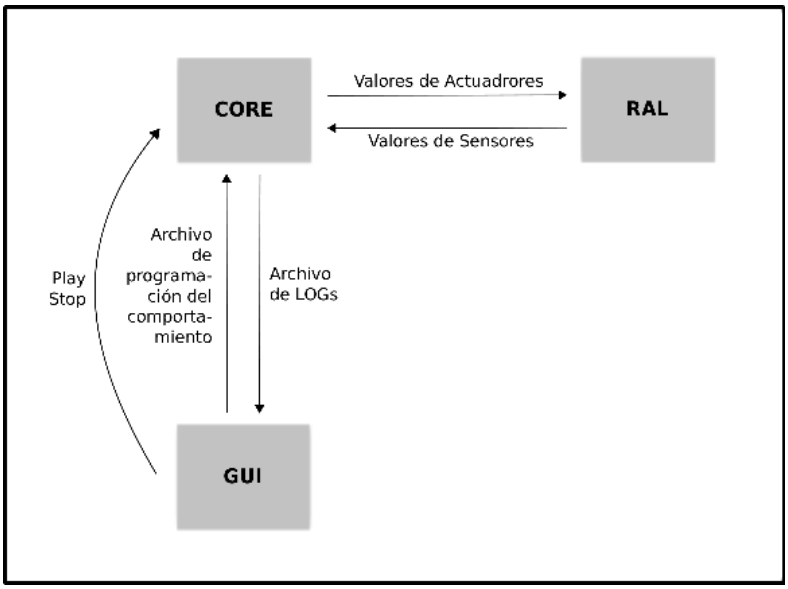

Figure 3.1: ERBPI ARQUITECTURA.

Chapter 4

Core

4.1 Introducción

The CORE module is in charge of executing the behaviour. It reads the XML behaviour file and establishes a connection with the appropriate RAL. At regular intervals, the core receives from the RAL the normalized values of the sensors, executes the behaviour, and gives to the RAL the normalized values to set the actuators. The CORE stops when the GUI signals the user has stopped the execution. To be able to execute the behaviour, the CORE has to transform the execution graph defined by the GUI in the behaviour-file to a corresponding set of ordered execution lists, one list for each node of the subsumption graph. The order is to guarantee that all the inputs for a function are ready when its turn to execute is up. To achieve this, we use a topological sorting [13] of the execution graph.

The CORE also performs different checkups to assure that the behaviour can be executed in the selected robot, for example that the graph is not cyclic (i.e, cannot be ordered) or that the robot has enough sensors and actuators to execute the behaviour. It also defines the communication frequency with the RAL depending on the robot, since each robot has a different working frequency. Finally, the CORE keeps a log-file where all the values at each tick are registered, including the execution time, each sensor value, the output value of each function and the value of each actuator. This log file is communicated to the GUI. We plan to use it to implement a debug function in the future.

El Core debería, a grandes rasgos, hacer las siguientes cosas:

1. **Parsear el XML:** Levantar el archivo XML, chequear que no haya elementos repetidos por *id*, chequear que no existan ciclos en el grafo formado por los sensores + cajas + actuadores, chequear que los predecesores de cada elemento sean elementos existentes de modo que el grafo sea consistente, chequear que las actualizaciones de las transiciones correspondan a elementos existentes, realizar un *topological sorting*[1] del grafo, y por último, devolver un objeto de tipo *Conducta* con el cual se realizará toda la ejecución posterior.

2. **Chequear que los sensores y actuadores del Core y el RAL se correspondan entre sí:** Obtener la lista de sensores y actuadores del *RAL* y realizar el chequeo de que el *RAL* contenga los sensores y actuadores que tiene el *Core*.

3. **Definir la frecuencia de trabajo:** Obtener la *frecuencia de trabajo* del *RAL* para luego ejecutar como máximo a esta *frecuencia*.

4. **Ejecutar:** Obtener el nuevo estado (valor) de los sensores del *RAL* y actualizar los valores en los elementos de la *Conducta* correspondientes. Para cada elemento del *comportamiento actual* actualizar su valor en función de sus predecesores, y por último, enviarle al *RAL* el nuevo valor para los actuadores y actualizar el archivo de *LOG* con el valor de todos los elementos la *Conducta*.

[1] *Topological Sort, Introduction to Algorithms*, Cormen, Leiserson, Rivest, and Stein.

Observaciones:

- En el parseo, no se chequea que una *Caja* no tenga como entrada a un *Actuador*.
- En el *LOG*, no se guarda el estado de sensores del *RAL*, se desprenden de la *Conducta*.
- En el chequeo entre sensores y actuadores del *Core* y el *RAL*, no se chequea que estén en el mismo orden, sólo que los sensores y actuadores del *Core* sean un subconjunto de los sensores y actuadores del *RAL*

4.2 Implementación

Lo hacemos en C++ para que sea lo más eficiente posible.

Para la funcionalidad de parseo del XML utilizamos el *Xerces XML Parser 3.0.1*[2] para C++. Recompilamos las librerías del Xerces de forma estática para que estén incluidas en el ejecutable del *Core* y no sea necesario transportalas.

El parseo del XML es bastante estricto y flexible al mismo tiempo. Sólo se obtienen del XML los datos necesarios para la ejecución del Core, cualquier otro atributo o especificación son ignorados. De esta forma, el XML podría contener información adicional, que el Core ignorará, pero que serviría para otros módulos como la *GUI*. Para más detalle sobre la estructura que el Core obtiene del XML, ver "XML - Datos para la Ejecución del Core" en el punto 7.2.1 en la página 39.

4.2.1 Estructuras

Para realizar el manejo en el *Core* de los sensores, cajas y actuadores, se utilizarán los siguientes objetos (C++ `class`):

```
clase Elemento (es una clase abstracta)
    atributos:
        string _id
        int _valor        // es el valor de salida
        int _entrada      // es la sumatoria de todas sus entradas
        int _tipo         // tipo: sensor, caja o actuador

    métodos:
        string getId()    // devuelve _id
        int setValor(int) // setea _valor y lo devuelve
        int getValor()    // devuelve _valor
        int getEntrada()  // devuelve _entrada
        int ejecutar()    // es virtual, llama a la de la clase hija...

clase Sensor (hereda de clase Elemento)
    atributos:
        // ninguno
    métodos:
        int ejecutar()    // devuelve valor de Elmemento::_valor

clase Caja (hereda de clase Elemento)
    atributos:
        vector<Elemento*> _entradas;
        vector<Punto> _puntos;
    métodos:
        int ejecutar()    // para cada "i" en _entradas, acumula *(_entradas[i]).getValor(),
                          // realiza resultado = función(acumulador), y luego, setea
```

[2] *Xerces XML Parser 3.0.1*, http://xerces.apache.org/xerces-c/

```
                            // el "resultado" en Elemento::_valor, setea el "acumulador"
                            // en Elemento::_entrada y devuelve Elemento::_valor

clase Actuador (hereda de clase Elemento)
    atributos:
        vector<Elemento*> _entradas;
    métodos:
        int ejecutar()      // para cada "i" en _entradas, acumula *(_entradas[i]).getValor()
                            // luego setea el acumulador en Elemento::_valor
                            // y devuelve Elmemento::_valor

struct Punto
  int x;
  int y;

string es la clase de C++ STL.
vector es la clase de C++ STL.
```

De esta forma, la *tabla de orden de ejecución secuencial* será un vector de la clase `Elemento`:

```
vector<Elemento> TablaEjecucion
```

Así, la ejecución sólo consistirá en recorrer la tabla secuencialmente y, por cada elemento, realizar el `ejecutar()` que se encargará de obtener los valores requeridos en $\mathcal{O}(1)$, ya que cada elemento contiene *punteros* a los elementos que le son predecesores: `TablaEjecucion[i].ejecutar()`

Observaciones:

- El método `Caja::ejecutar()`, debe tener en cuenta al calcular de manejar los datos en valores `float` para no perder precisión ni entrar en casos en los que devuelva *cero* por truncamiento a `int`.
- El método `Caja::ejecutar()`, asume que los 2 puntos están ordenados, es decir, `Caja._puntos[0].x` $\leqslant$ `Caja._puntos[1].x`. Por lo tanto, calcular el valor de la función se reduce a 3 casos:
  - $entrada \leqslant x_0 \implies resultado = y_0$
  - $entrada \geqslant x_1 \implies resultado = y_1$
  - $x_0 < entrada < x_1 \implies resultado = \frac{y_0 - y_1}{x_0 - x_1} \times (entrada - x_0) + y_0$ (ecuación de la recta)
- Por como están diseñadas las clases, y posteriormente los algoritmos, obliga a que los atributos `_entradas` y `_puntos` sean públicos. De lo contrario, habría que especificarlos como privados y especificar sus métodos correspondientes. Que sean atributos públicos y no tengan sus métodos correspondientes, hace que cualquier función pueda modificar a su antojo cualquier atributo de la clase, y que, en los algoritmos como el Parser, para crear los elementos haya que agregar las `&(Elemento)` en `_entradas` y los puntos en `_puntos` manualmente...

4.2.2 Pseudocódigos

4.2.2.1 *Core*

Preprocesamiento:

vector<Elemento> TablaEjecucion ⟵ Parsear(ArchivoXML, TablaEjecucion)
if (ids sensores en TablaEjecucion) $\nsubseteq$ (ids RAL.getListaSensores()) **then**
  Error: los sensores no se corresponden y Terminar
if (ids actuadores en TablaEjecucion) $\nsubseteq$ (ids RAL.getListaActuadores()) **then**
  Error: los actuadores no se corresponden y Terminar
frecuencia ⟵ RAL.getFrecuenciaTrabajo()
Ejecución:
while frecuencia lo permita **do**
  vector<<id;valor>> sensoresRAL ⟵ RAL.getEstadoSensores()
  actualizar el nuevo valor de cada sensor en TablaEjecucion con sensoresRAL
  for cada elemento i de TablaEjecucion **do**
    TablaEjecucion[i].ejecutar()
  vector<<id;valor>> actuadoresRAL ⟵ generado con los actuadores de la TablaEjecucion
  RAL.setEstadoActuadores(actuadoresRAL)
  LOG ⟵ actualizar con TablaEjecucion

4.2.2.2 *Parsear(in ArchivoXML, inout TablaEjecucion)*

vector<<tipo:char; id:string; entradas:vector<string>; puntos:vector<<int;int>>>>
vectorAuxiliar ⟵ generado con cada elemento (sensor, caja o actuador) parseado de ArchivoXML
for cada elemento i en vectorAuxiliar **do**
  if $\exists j, j \neq i$ / vectorAuxiliar[j].id = vectorAuxiliar[i].id **then**
    Error: hay IDs repetidos y Terminar
for cada elemento i en vectorAuxiliar **do**
  for cada elemento j en vectorAuxiliar[i].entradas **do**
    if $\nexists k, 0 \leqslant k < long(vectorAuxiliar)$ / vectorAuxiliar[k].id = vectorAuxiliar[i].entradas[j] **then**
      Error: hay elementos que tienen "entradas" que no existen y Terminar
if HayCiclos(vectorAuxiliar) **then**
  Error: el grafo contiene ciclos y Terminar
vectorAuxiliar ⟵ TopologicalSorting(vectorAuxiliar)
vector<Elemento> TablaEjecucion ⟵ vacío
for cada elemento i en vectorAuxiliar **do**
  if vectorAuxiliar[i].tipo = Sensor **then**
    nuevo sensor(vectorAuxiliar[i].id)
    sensor.setValor(0)
    agrego el sensor en TablaEjecucion al final
  if vectorAuxiliar[i].tipo = Caja **then**
    nuevo caja(vectorAuxiliar[i].id)
    for cada elemento j en vectorAuxiliar[i].puntos **do**
      agrego vectorAuxiliar[i].puntos[j] en caja._puntos al final
    for cada elemento j en vectorAuxiliar[i].entradas **do**
      for cada elemento k en TablaEjecucion $(0 \leqslant k \leqslant i)$ **do**
        if vectorAuxiliar[i].entradas[j] = TablaEjecucion[k].getId() **then**
          agrego &(TablaEjecucion[k]) en caja._entradas al final
    agrego la caja en TablaEjecucion al final
  if vectorAuxiliar[i].tipo = Actuador **then**
    nuevo actuador(vectorAuxiliar[i].id)

for cada elemento j en vectorAuxiliar[i].entradas **do**
for cada elemento k en TablaEjecucion ($0 \leqslant k \leqslant i$) **do**
if vectorAuxiliar[i].entradas[j] = TablaEjecucion[k].getId() **then**
agrego &(TablaEjecucion[k]) en actuador._entradas al final
agrego el actuador en TablaEjecucion al final
return TablaEjecucion

Observaciones: Por el momento, al generar el `vectorAuxiliar` se chequea que la *caja* tenga definidos exactamente 2 puntos, de lo contrario, termina con *ERROR*.

4.3 Ejecución

4.3.1 Linux 32 bits

No es necesario ejecutar el Core manualmente, de esto se encarga la GUI.

4.3.1.1 Script

De todas formas, si se quisera probar manualmente su ejecución, puede utilizarse el script *ERBPI/src/core/core_ ejecutar.sh*, editándolo y modificándolo con los parámetros requeridos.

Para ejecutar el *Core* se deberán especificar 3 parámetros:

1. **ArchivoXML:** El archivo *XML* a parsear.
2. **ArchivoLOG:** El archivo de *LOG* donde se guardará el *log* de la ejecución.
3. **RAL_ID:** la especificación del *RAL* que se utilizará.

Por ejemplo: `./core ArchivoXML.xml ArchivoLOG.log RAL_ID`
donde `RAL_ID` podría ser: *exabot*, *khepera*, *yaks*, etc.

4.3.1.2 Ejecución del Core junto con el RAL

Como ya dijimos, el *Core* se encuentra compilado con una *librería dinámica* del *RAL*. Por lo tanto, es necesario indicarle al *sistema operativo* dónde buscar la librería dinámica `libRAL.so` cuando el *Core* llame a funciones de la misma. De lo contrario, la ejecución falla.

La forma de hacer esto en *Linux* es, en la misma consola donde se ejecutará el *Core*, ejecutar las siguientes dos líneas para agregar al sistema operativo un *path* para la búsqueda de librerías:

```
# LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<path>
# export LD_LIBRARY_PATH
```

donde <*path*> debe ser la ruta (absoluta) donde se encuentra la librería dinámica *libRAL.so*, por ejemplo:

```
# LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/usuario/desktop/soft_src/ral/src
```

También podría modificarse el archivo de configuración del usuario `.profile` para agregar esta ruta de forma permanente. Para más información sobre el manejo de librerías dinámicas en *Linux*, puede consultarse http://www.chuidiang.com/clinux/herramientas/librerias.php

IMPORTANTE !!: Por ahora las librerías del RAL se llaman todas iguales libRAL.so y vamos pisando con la que corresponde en la carpeta de ejecución del Core... Luego, hay que hacer un "if" en el Core, para que cargue en tiempo de ejecución la librería que corresponda (*libRAL-yaks.so* o *libRAL-exabot.so*). También va a ser necesario tocar unas cositas en el RAL para que esto quede bien.

4.3.2 Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

4.4 Finalización

4.4.1 Linux 32 bits

Al comenzar a ejecutar el *Core*, el mismo entra en un *ciclo infinito* en el que va ejecutando y actualizando los valores de todos los elementos.

Para terminar la ejecución del *Core*, el mismo tiene definida un *rutina de atención de señales*, en particular para las señales `SIGINT` y `SIGTERM`, que al ser recibida por el *Core* termina su ejecución de forma ordenada, deteniendo los motores por seguridad enviandoles valor 0 (cero), finalizando los procesos correpondientes, destruyendo las esctructuras dinámicas y cerrando correctamente el archivo de *log*.

Las señales se encuentran definidas en la librería estandar `<signal.h>`. Puede verse la especificación de señales en [7].

La señal `SIGINT` (*interrupt key signal*), es una señal de atención interactiva, generalmente generada por la teclas `Ctrl+C` en la consola de ejecución, pero que también puede ser enviada por otro programa.

La señal `SIGTERM` (*termination signal*), es una señal de terminación enviada por el comando `kill`, pero que también puede ser enviada por otro programa.

La idea es que sea la *GUI* la que inicia la ejecución del *Core* y la que termine la ejecución del mismo enviando la señal `SIGINT` o `SIGTERM` según corresponda.

Nota: Es muy importante el orden en el que se llama a las distintas funciones, por ejemplo `inicializarRAL()`, `signal(SIGINT,terminar)` y `signal(SIGTERM,terminar)`, ya que esto determinará qué procesos afectará la atención de señales. Por ejemplo, teníamos un bug en la finalización por señales al utilizar RAL-ExaBot, ver en 11.2.5 y 11.2.6.

4.4.2 Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

4.5 Compilación

El código fuente del Core cuenta con los siguientes archivos:

1. **core.cpp** Código principal para la ejecución del *Core*.

2. **Estructuras.h** Encabezados de las clases utilizadas para la ejecución del *Core*.
3. **Estructuras.cpp** Código de las clases utilizadas para la ejecución del *Core*.
4. **Parser.h** Encabezados de las funciones para el parseo del archivo *XML*.
5. **Parser.cpp** Código de las funciones para el parseo del archivo *XML*.

Además, el código fuente del Core necesita para su compilación los siguientes archivos del *RAL*:

1. **RAL.h** Encabezados de las funciones del *RAL*.
2. **libRAL.so** Librería dinámica del *RAL*.

Puede verse la especificación para la compilación de estos archivos del *RAL* en el punto 5.4.1 en la página 16.

4.5.1 Linux 32 bits

4.5.1.1 Makefile

El código fuente del Core incluye un archivo *Makefile* con las siguientes funciones para facilitar la compilación, linkeo estático con el *Xerces Parser*, linkeo dinámico con el *RAL* y el testeo del Core:

1. **all:** Ejecuta las funciones *compilar_core* y *enlazar_ejecutable*.
2. **compilar_core:** Compila los archivos del código fuente del *Core* generando los objetos (**.o*) necesarios para la creación del ejecutable del Core de la siguiente manera:
   g++ -c Estructuras.cpp -o Estructuras.o
   g++ -c Parser.cpp -o Parser.o
   g++ -c core.cpp -o core.o
3. **enlazar_ejecutable:** Genera el ejecutable del *Core* (`test_core`) enlazando con las *librerías estáticas* del *Xerces* y con las *librerías dinámicas* del *RAL* de la siguiente manera:
   g++ -o test_core core.o Estructuras.o Parser.o -lxerces-c -lpthread -L<path> -Bdynamic -lRAL
   donde *<path>* debe ser la ruta (puede ser relativa) donde se encuentra la librería dinámica *libRAL.so*, por ejemplo `-L../../ral/src`
4. **clean:** Borra todos los archivos **.o* y el ejecutable *test_core*.
5. **run:** Ejecuta *test_core* con los siguientes parámetros:
   ./test_core xml_file_test_07.xml archivoLOG_01.log yaks

4.5.1.2 Script

Para facilitar algunas cuestiones en la compilacion, también se incluye un archivo script "core_compilar.sh". El mismo puede ejecutar simplemente con `./core_compilar.sh`.

4.5.2 Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

4.6 Xerces XML Parser

ver Capítulo 9 en página 49

4.7 Instalación

4.7.1 Linux 32 bits

Para la instalación se ejecuta el script *ERBPI/src/core/core_instalar.sh*. Es necesario que el Core y las librerías dinámicas de cada RAL ya se encuentren compiladas con anterioridad. Para compilación de Core y RAL ver puntos 4.5 y 5.4.

4.7.2 Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

4.8 Compiladores

4.8.1 C++ Linux 32 bits

Utilizamos los siguientes compiladores para C++:

- gcc: GCC (GNU Compiler Collection) C compiler.
- g++: GCC (GNU Compiler Collection) C++ compiler.

Ver `http://gcc.gnu.org/`

4.8.2 C++ Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! Para que los comandos como *gcc*, *g++*, *make* (*mingw32-make*) anden en la consola de windows, es necesario modificar la variable de sistema `PATH` de Windows y agregar la ruta `C:/MinGW/bin`.

FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de `http://www.cygwin.com/`. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

Chapter 5

RAL: Robot Abstraction Layer

5.1 Introducción

The RAL modules encapsulates all the knowledge of the particular robot or simulator, providing a standard interface to the CORE module, and dealing with everything necessary to communicate with the actual robot. The RAL abstracts the particular robot, its communication protocol, and normalizes the values of the particular sensors and actuators. In this way, all the specific characteristics of the robot are transparent to the CORE: the RAL provides a standard interface that allows the CORE to get the list of sensors and actuators in the robot, the frequency the robot can work in, the normalized sensor values, and set the normalized values for the actuators.

To add a new robotic platform for ERBPI to work with, a programmer must only program a particular RAL for the platform implementing the general RAL interface. All RALs are implemented as dynamic libraries. In this manner, we can add new RALs without having to recompile the CORE or the GUI. Moreover, this allows the CORE to load a different RAL on runtime, without having to restart the application. This makes ERBPI easily extendable to control different robots.

La idea es que sea una capa de abstraction respecto del hardware específico que hay del otro lado, es decir, qué tipo de robot, simulador, qué tipo y cantidad de sensores y actuadores, etc. Por lo tanto, para el *Core* va a ser transparente, sólo se comunicará con el *RAL* para recibir el estado de los sensores y enviar el nuevo estado para los actuadores. Luego, será el *RAL* el que se comunicará directamente con el hardware o simulador según corresponda (Khepera, ExaBot, Yaks, etc.).

Debería hacer las siguientes cosas:

- **getListaSensores().** Devolver una lista de IDs de los sensores que posee el hardware o simulador que se se está utilizando.
- **getListaActuadores().** Devolver una lista de IDs de los actuadores que posee el hardware o simulador que se se está utilizando.
- **getEstadoSensores().** Devolver una lista de <id;valor> con el nuevo estado de cada sensor del hardware o simulador que se se está utilizando.
- **getFrecuenciaTrabajo().** Devolver a qué frecuencia sensa y es posible asignarle a los actuadores el hardware o simulador que se se está utilizando, para que el *Core* lo tenga en cuenta y trabaje a esta frecuencia como máximo...
- **setEstadoActuadores().** Recibir una lista de <id;valor> con el nuevo valor para cada actuador y actualizar los actuadores en el hardware o simulador que se se está utilizando.
- **inicializarRAL().** Inicializar el hardware o simulador que se se está utilizando.
- **finalizarRAL().** Finalizar el hardware o simulador que se se está utilizando.

- **Comunicación con el Hardware.** Realizar la conexión por software con el hardware específico o simulador que se utilizará y enviar los comandos correspondientes para que se mueva...

5.2 Implementación

Lo hacemos en `C++` como una *librería dinámica multiplataforma* (.DLL o .SO) para interactuar directamente con el *Core*, sin la necesidad de recompilar el *Core* para distintos *RALs*. Luego, para interactuar con otro robot o simulador, simplemente se le especificará por línea de comandos al *Core* cuál será el *RAL_ID* que se utilizará.

Por lo tanto, la *librería diámica del RAL* será una sola, y el mismo *RAL* deberá poder diferenciar sobre qué hardware deberá trabajar... ¿ESTO LO HACEMOS CON UN PARÁMETRO? ¿ESTE PARÁMETRO DEBERÍA IR EN C/U DE LAS FUNCIONES DEL RAL? RESOLVER ESTO...

Por el momento, la idea es tener las siguientes *RALs*:

To the date, we have implemented RALs for the Khepera [14] and Exabot [15] robots, and for the YAKS (Yet another Khepera Simulator) [16] and the Player/Stage [17] simulator adapted for the ExaBot.

5.2.1 Normalización de Valores

El módulo RAL se encarga además de normalizar los valores de sensores y actuadores. De esta forma, se abstrae para el módulo GUI cómo es el manejo e interpretación de los valores de cada sensor y actuador dependiendo del robot o simulador que se esté utilizando.

Para esto, el módulo GUI sólo maneja valores relativos de los sensores y actuadores. Es decir, para sensores maneja valores en el rango `[0:100]` que indican el porcentaje del valor del sensor (`[0%:100%]`). Para motores, maneja valores en el rango `[-100:100]` que indican el porcentaje del valor del motor (`[-100%:100%]`). El módulo RAL recibe del módulo GUI estos valores relativos (normalizados), se encarga de desnormalizarlos adecuadamente en función del robot o simulador que se esté utilizando, y enviar los valores desnormalizados al robot. Cuando el módulo RAL recibe valores del robot, se encarga de normalizarlos (relativizarlos) antes de entregar estos valores al módulo GUI.

Para más información de la normalización específica para cada robot y simulador, ver apartados 5.7.2.1, 5.8.2.2, 5.9.2.1 y 5.10.2.1.

5.3 Ejecución

la llama dinámicamente el Core

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

5.4 Compilación

5.4.1 Linux 32 bits - Librería Dinámica

El código fuente del *RAL* cuenta con los siguientes archivos:

1. **RAL.h** Encabezados de las funciones para la utilización de la librería dinámica del *RAL*.

2. **RAL.cpp** Código de las funciones de la librería dinámica del *RAL*.

5.4.1.1 Makefile

Además, el código fuente incluye un archivo *Makefile* con las siguientes funciones para facilitar la compilación de la librería dinámica del *RAL*:

1. **all:** Compila y enlaza los archivos del código fuente generando la librería dinámica *libRAL.so* de la siguiente manera:
   g++ -c RAL.cpp -o RAL.o
   ld -o libRAL.so RAL.o -shared ó *g++ -shared -Wl -o libRAL.so RAL.o* (dependiendo el caso)

2. **clean:** Borra todos los archivos **.o* y **.so* del *RAL*.

Para más información sobre la creación, compilación y enlace de librerías dinámicas en *Linux*, puede consultarse http://www.chuidiang.com/clinux/herramientas/librerias.php

5.4.1.2 Error de Compilación en 64 bits

Si el Linux es de 64 bits, es probable que falle la compilación de *RAL.cpp*. La solución es agregar en la línea de compilación el parámetro "`-fPIC`", de forma que la línea antes indicada quede como "*g++ -c RAL.cpp -o RAL.o -fPIC*".

5.4.2 Windows 32 bits - Librería Dinámica

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

5.5 Instalación

lo hace la instalacion del Core!!!

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

5.6 Compiladores

5.6.1 C++ Linux 32 bits

Utilizamos los siguientes compiladores para C++:

- gcc: GCC (GNU Compiler Collection) C compiler.
- g++: GCC (GNU Compiler Collection) C++ compiler.

Ver http://gcc.gnu.org/

5.6.2 C++ Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! Para que los comandos como *gcc*, *g++*, *make* (*mingw32-make*) anden en la consola de windows, es necesario modificar la variable de sistema `PATH` de Windows y agregar la ruta `C:/MinGW/bin`.

FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

5.7 RAL Khepera

5.7.1 Khepera

Khepera es un robot móvil desarrollado por la empresa K-Team. Tiene un cuerpo circular, de 5,5cm de diámetro, y consta de dos ruedas (actuadores) y ocho pares de sensores infrarojos, que pueden funcionar como sensores de proximidad o de luz direccionales. Estos robots pueden ser controlados desde una PC a través de una interface serie o de una interface de radio.

Para más información ver [3], [4], [5] y [6].

5.7.2 Implementación

En función de las especificaciones del robot *Khepera* y las características particulares necesarias para la conexión con el mismo, ya sea a través de *Cable Serial* o *Radio Frecuencia*, realizamos dos *RALs* de *Khepera*, una para Linux y otra para Windows.

5.7.2.1 Normalización de Sensores y Actuadores

Los valores absolutos (desnormalizados) y normalizados que maneja el Khepera para sensores y actuadores son:

componente	valor mínimo	normalizado	valor máximo	normalizado
sensor proximidad	0	0%	1023	100%
sensor luz	0	0%	512	100%
motores	-20	-100%	20	100%

La función de normalización-desnormalización se encuentra implementada en las funciones `normalizarSensores()` y `desNormalizarMotores()`. Estas funciones simplemente son una conversión lineal (regla de tres simple) entre el valor del sensor o motor y los valores máximos y mínimos. Una vez normalizados o desnormalizados los valores, los mismos son saturados a los valores máximos y mínimos permitidos para evitar problemas tanto en el robot como en la GUI. Los valores absolutos posibles para los motores son limitados, aunque el robot Khepera admite valores mayores, no se deben superar los indicados en el rango `[-20:20]` para cuidar la mecánica del robot. Para más información sobre este punto ver [4](página 25).

FALTA PONER QUÉ QUIERE DECIR EL VALOR EN MILIMETROS DE `[0:1023]` Y `[0:512]` PARA LOS SENSORES !!!!!
FALTA PONER QUÉ QUIERE DECIR EL VALOR EN MILIMETROS DE `[0:1023]` Y `[0:512]` PARA LOS SENSORES !!!!!
FALTA PONER QUÉ QUIERE DECIR EL VALOR EN MILIMETROS DE `[0:1023]` Y `[0:512]` PARA LOS SENSORES !!!!!

Nota: tener en cuenta al hacer los cálculos de normalización-desnormalización que deben manejarse los datos en valores `float` para no perder precisión ni entrar en casos en los que devuelva *cero* por truncamiento a `int`.

5.7.3 Conexión por Cable

Primero es importante chequear que el robot esté correctamente configurado para este tipo de conexión. En la sección *3.1.3 - Jumpers, reset button and settings* del manual de usuario se detalla los modos de conexión. La configuración correcta es `MODE 1`, que implica una conexión *serial RS232 a 9600 Baud*. Ver Figura 5.1.

5.7.3.1 Configuración Serial RS232

Para saber cómo es el protocolo de comunicación serial ver manual Khepera página 18, punto *6. The serial communication protocol.*

Cuidado: No hay que tener la torreta puesta y tiene que estar bien configurado el modo de RS232, ver Figura 5.1.

La conexion desde la PC-host siempre es por RS232 y configurada a "*XXXX Baud, 8 bit, 1 start bit, 2 stop bit, no parity*", lo único que puede cambiar son los *XXXX Baud*, que los configuramos a "*9600 Baud*", entonces nuestra configuración de *RS232*:

- Port /dev/ttyS0 (para linux)
- 9600 Baud
- 8 Bit
- 1 StartBit
- 2 StopBit
- NONE Parity
- RTS/CTS FlowControl

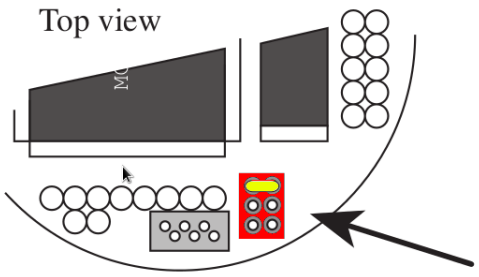

Figure 5.1: Configuración de jumpers para el control del robot en "*Modo 1*" para el protocolo de comunicación serial a 9600 Baudios.

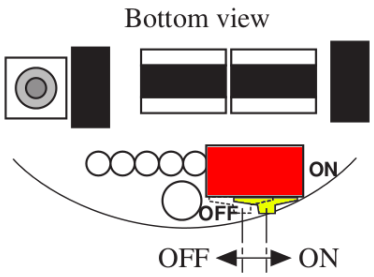

Figure 5.2: Configuración de jumpers para el control de la batería del robot.

5.7.3.2 Alimentación del robot

Si el switch de las baterias está en **ON:** el robot se power-alimenta de las baterias internas. Si está en **OFF:** el robot se power-alimenta con el cable serial de datos (6 pines). Ver Figura 5.2. Ver manual Khepera, página 15, punto 5.2 *Configuration for robot-computer communication* dice: "*Between the robot and the interface/charger module by the S serial cable. This cable also supports the power supply of the robot. This external power supply is available when the general battery switch is OFF. If the switch is ON, the robot uses its own batteries for power supply.*"

5.7.4 Conexión por Radio

No cambia nada, es lo mismo comunicarse con el robot "con cable" y con "torreta-radio". Para usar con "torreta-radio" sólo hace falta agregar (enviar) el comando "`*ID_destination\n`", donde `ID_destination` podrían ser distintos valores en función de estar utilizando simultáneamente distintos robots. En nuestro caso, usamos por defecto el Robot Nº 1 por lo que el comando debe ser "`*1\n`" antes de empezar la transmisión para que el "radio base" sepa a qué robot mandar. Además, la torreta del khepera en el robot tiene que estar switcheada en "1" para aceptar los comandos, o sea, sólo trabajaremos con todo configurado en Robot Nº 1. Para simplidicar y unificar las RALs (cable y radio) simplemente agregamos siempre el comando "`*1\n`", que sirve para *radio* y es ignorado para *cable*.

5.7.4.1 Configuración Torreta-Radio

Para usar el robot con *torreta-radio*, además de montarla sobre el robot, hay que configurar la misma. Ver Figura 5.3. Ver manual Khepera Radio Turret User, página 6, punto 4.5

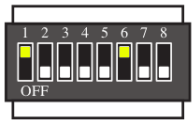

Figure 5.3: Configuración de jumpers para la torreta-radio del robot.

Running mode and ID selector. Switches 1 to 6 are used to specify running mode and the radio turret ID. Switches 7 and 8 are not used and have to be always set to 0. Switches 1 to 5 define the 5 bits of the turret ID. Switch 6 defines the running mode of the turret: when this switch is in the OFF position, the turret is a normal extension turret. When this switch is in the ON position, the turret is used as main communication channel of the module COM. We set Switch 1 y 6 to ON, y los switchs 2, 3, 4, 5, 7 y 8 to OFF. The radio turret ID should be never set to 0.

5.7.5 RAL Khepera Linux 32 bits

En Linux, para controlar el puerto serial, o más conocido como `COM1`, se realiza a través del archivo de sistema `/dev/ttyS0`, donde en general:

- `/dev/ttyS0` ó `/dev/cua0` corresponde con el puerto `COM1` en Windows
- `/dev/ttyS1` ó `/dev/cua1` corresponde con el puerto `COM2` en Windows
- `/dev/ttyS2` ó `/dev/cua2` corresponde con el puerto `COM3` en Windows
- `/dev/ttyS3` ó `/dev/cua3` corresponde con el puerto `COM4` en Windows

Por lo tanto en `C++`, generar una conexión a través del puerto serial, leer, escribir y cerrar el mismo se realiza con las funciones comunes para manejo de *streams* y archivos:

```
#include <iostream>
open( "/dev/ttyS0" );
close( file_descriptor );
write( file_descriptor );
read( file_descriptor );
```

Es importante el modo en el que se abre el archivo del `COM1` y más importante, configurar el puerto para la conexión necesaria para el robot Khepera. Esto se hace de la sigueinte manera:

```
// se abre el COM1 en modo "non-blocking"
com1_file_descriptor = open( "/dev/ttyS0", O_RDWR | O_NOCTTY | O_NONBLOCK );
        // O_RDWR - open read-write.
        // O_NOCTTY - open TTY without it becoming controlling tty.
        // O_NONBLOCK ó O_NDELAY - open in non-blocking mode (read will return immediately)
```

Luego se configura el puerto antes de comenzar a utilizarlo:

```
// se definen constantes para simplificar
#define BAUDRATE    B9600   // BAUDRATE = 9600
#define DATABITS_8  CS8     // DATABITS = 8 bits
#define STOPBITS_2  CSTOPB  // STOPBITS_2 = 2
#define PARITYON    0       // es igual a PARITY_NONE ó PARITY_DISABLED
#define PARITY      0       // es igual a PARITY_NONE ó PARITY_DISABLED
// se crea la estructura para setear la configuración del puerto
struct termios com1_new_set;
com1_new_set.c_cflag = ( BAUDRATE | CRTSCTS | DATABITS_8 | STOPBITS_2 | PARITYON | PARITY | CLOCAL | CREAD );
```

```
com1_new_set.c_iflag = IGNPAR;
com1_new_set.c_oflag = 0;
com1_new_set.c_lflag = 0;
com1_new_set.c_cc[VMIN] = 1;
com1_new_set.c_cc[VTIME] = 0;
tcflush( com1_file_descriptor, TCIFLUSH );
// se setea la nueva configuración para el puerto COM1
tcsetattr( com1_file_descriptor, TCSANOW, &com1_new_set );
```

La compilación de este *RAL* es básicamente la misma a la de todos los demás, con la diferencia que el proceso de *linkeo* para generar la librería dínamica debió ser levemente cambiado. El error ocurría al intentar *linkear* el ejecutable del *Core* dinámicamente con *libRAL.so*, producía el siguiente error:

```
hidden symbol `__dso_handle' in /usr/lib/gcc/i486-linux-gnu/4.3.3/crtbegin.o is referenced by DSO
/usr/bin/ld: final link failed: Nonrepresentable section on output
collect2: ld returned 1 exit status
```

Por lo tanto, en el *Makefile* incluído en los archivos fuentes de este *RAL*, para generar la librería dinámica *libRAL.so* el proceso de *linkeo* que se realizaba mediante *ld -o libRAL.so RAL.o -shared* fue cambiado por ***g++ -shared -Wl -o libRAL.so RAL.o***.

Por último, la frecuencia de trabajo que devuelve esta librería (*getFrecuenciaTrabajo()*) es igual al valor que devuelve el *RAL YAKS* (100*mseg*). Según las pruebas que se realizaron parece andar bien, pero si aparecieran inconvenientes será necesario revisar este valor. Tener en cuenta que la frecuencia de trabajo debería quedar determinada por:

- Envío de comando y tiempo de transmisión de esa cantidad de caracteres a 9600 baudios.
- Tiempo de sensado (de todos los sensores ¿16?) del *Khepera*.
- Tiempo de transmisión de las cantidad de caracteres de la respuesta a 9600 baudios.
- Sumatoria de todo lo anterior...

Para probar el robot manualmente (chequear que se tiene conexión con el mismo), se pueden instalar y usar los siguientes programas para el manejo del puerto (como el *hyperterminal* de Windows):

```
sudo apt-get install gtkterm
sudo apt-get install setserial
```

5.7.5.1 BUGs

Hay un bug en la ejecución, ver 11.1.1

Hay un problema con el RAL Khepera-Torreta-Radio, no anda!!! Ver 11.1.2

5.7.6 RAL Khepera Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

5.8 RAL YAKS

5.8.1 YAKS

YAKS es un simulador de cóìdigo abierto, escrito en `C++`, de robots tipo `Khepera`, desarrollado por Johan Carlsson. Su nombre proviene del acrónimo *Yet Another Khepera Simulator*. Posee las siguientes características:

- Permite incluir en el entorno obstáculos circulares, paredes, luces y definir zonas.
- Permite definir y manipular un número ilimitado de robots.
- Permite separar el programa de control del simulador, ya que los robots pueden ser manejados a través de una conexión `TCP/IP`.
- Soporta una gran variedad de sensores: proximidad, luminosidad, energía, encoders de las ruedas, compás y sensor de tierra (para detección de zonas).

PONER UNA IMAGEN DEL YAKS!!!!!

5.8.2 Implementación

5.8.2.1 Bug

Había un bug en el valor que se seteaba a los motores a diferencia del Khepera. Ver en 11.2.4.

5.8.2.2 Normalización de Sensores y Actuadores

Los valores absolutos (desnormalizados) y normalizados que maneja el Yaks para sensores y actuadores son:

componente	valor mínimo	normalizado	valor máximo	normalizado
sensor proximidad	0	0%	1023	100%
sensor luz	0	0%	512	100%
motores	-20	-100%	20	100%

La función de normalización-desnormalización se encuentra implementada en las funciones `normalizarSensores()` y `desNormalizarMotores()`. Estas funciones simplemente son una conversión lineal (regla de tres simple) entre el valor del sensor o motor y los valores máximos y mínimos. Una vez normalizados o desnormalizados los valores, los mismos son saturados a los valores máximos y mínimos permitidos para evitar problemas tanto en el robot como en la GUI.

Hay que tener mucho cuidado con los valores absolutos que se envían al Yaks para los motores. Por alguna razón de la implementación del Yaks, si se superan los valores del rango `[-9:10]` el Yaks satura a `0` (cero) el valor de los motores y el robot no se mueve. A pesar de esto, por una cuestión de compatibilidad Yaks-Khepera en la GUI, mantuvimos en el Yaks los valores absolutos de motores del Khepera para la desnormalización. Luego de esto, saturamos los valores absolutos antes de enviárselos al robot dentro del rango `[-9:10]`, esto hace que en el Yaks, los motores nunca superen el rango relativo `[-50%:50%]`.

FALTA PONER QUÉ QUIERE DECIR EL VALOR EN MILIMETROS DE `[0:1023]` Y `[0:512]` PARA LOS SENSORES !!!!!
FALTA PONER QUÉ QUIERE DECIR EL VALOR EN MILIMETROS DE `[0:1023]` Y `[0:512]` PARA LOS SENSORES !!!!!
FALTA PONER QUÉ QUIERE DECIR EL VALOR EN MILIMETROS DE `[0:1023]` Y

[0:512] PARA LOS SENSORES !!!!!

Nota: tener en cuenta al hacer los cálculos de normalización-desnormalización que deben manejarse los datos en valores `float` para no perder precisión ni entrar en casos en los que devuelva *cero* por truncamiento a `int`.

5.9 RAL ExaBot

5.9.1 ExaBot

5.9.2 Implementación

Nota: Falta hacer una función `inicializarRAL(lista sensores)`, ver bug 11.1.5.

5.9.2.1 Normalización de Sensores y Actuadores

Los valores absolutos (desnormalizados) y normalizados que maneja el Exabot para sensores y actuadores son:

componente	valor normalizado mínimo		valor normalizado máximo	
sensor infrarrojo de proximidad (telémetro)	39	0%	157	100%
sensor infrarrojo de línea (line-following)	[2:255]	0%	1	100%
sensor contacto (bumper)	0	0%	255	100%
sensor radio (sonar)	31250	0%	125	100%
motores	-30	-100%	30	100%

La función de normalización-desnormalización se encuentra implementada en las funciones `linealizar()`, `normalizarLinea()`, `normalizarBumper()`, `normalizarSonar()` y `setEstadoActuadores()`. A continuación, y en función de cómo está configurado el hardware del robot en cada caso, describimos cada una de ellas:

linealizar(): Esta función corresponde a la normalización de los sensores infrarrojos de proximidad (telémetro).

Estos sensores arrojan valores entre `[0:159]`. Este rango de valores no se corresponde con una función lineal, por lo que será necesario posteriormente linealizar estos valores. De esta forma el primer paso en la normalización es traducir el valor absoluto del sensor en un valor de distancia en milímetros, donde $0 = 800mm$ y $159 = 60mm$. Notar que al traducir estos valores a distancia, se invierte la relación de máximo y mínimo, ahora el valor `60mm` es el máximo, donde el telémetro está viendo mucho o un objeto muy cercano, y el valor `800mm` es el mínimo, donde el telémetro está viendo poco o un objeto muy lejano.

Para linealizar a distancia los valores de estos sensores no se cuenta con una función. Para esto, se ha desarrollado una tabla de conversión sobre la base de experimentos que indican con qué valores de distancia en milímetros se corresponde cada valor del rango `[0:159]`.

Una vez linealizado el sensor, saturamos los valores. El sensor se encuentra dentro del robot a 70mm del borde del chasis, por lo que a partir de aquí tomaremos nuestra referencia del objeto más cercano, un objecto a 0mm del chasis, es decir, el valor normalizado `100%`. Por otro lado, si bien el sensor tiene la capacidad de ver objetos hasta los 800mm, sobre la base de experimentos determinamos más práctico limitar el sensor a los 350mm.

Entonces los valores del sensor son saturados en el rango `[39:157]`, que equivale linealmente a `[350mm:70mm]` y relativamente (normalizado) a `[0%:100%]`.

Una vez obtenido linealmente en distancia y saturado el valor del sensor, procedemos a normalizar esta distancia teniendo en cuenta que se encuentran invertidos el máximo y mínimo. Para esto simplemente realizamos una conversión lineal (regla de tres simple) entre el valor del sensor y los valores máximos y mínimos invertidos. Una vez normalizados estos valores, pueden ser entregados el módulo GUI en el rango `[0%:100%]`.

normalizarLinea(): Esta función corresponde a la normalización de los sensores infrarrojos de línea (line-following) para detección de línea blanca en el piso.

Estos sensores arrojan valores entre `[1:255]` indicando el tiempo transcurrido desde la última vez que visualizó la línea, donde $255 = \infty$ *tiempo*. En función de esto, el primer paso será interpretar estos valores absolutos.

Por una cuestión práctica, en esta función de normalización, interpretaremos el valor absoluto `1` como que se está viendo la línea en ese preciso momento y los valores absolutos en el rango `[2:255]` como que no se está viendo la línea.

Como último paso, normalizamos los valores absolutos a $1 = 100\%$ y cualquier otro valor a 0% (valor $\neq 1 \Rightarrow 0\%$)

Tener en cuenta que esta iterpretación del tiempo devuelto por el sensor podría cambiarse para obtener linealmente valores de tiempo en milisegundos indicando cuándo fue la última vez que se visualizó la línea.

normalizarBumper(): Esta función corresponde a la normalización de los sensores de contacto (bumper).

Estos sensores sólo arrojan dos valores `0` y `255`. El valor `0` indica que el sensor está libre (no apretado) y el valor `255` indica que el sensor está apretado. No existe rango de valores. Por lo que la normalización simplemente consiste en que los dos valor absolutos posibles se traducen como $0 = 0\%$ y $255 = 100\%$

normalizarSonar(): Esta función corresponde a la normalización del sensor de radio (sonar).

Este sensor arroja valores entre `[125:31250]` indicando el tiempo transcurrido desde la última vez que visualizó un objeto. Estos valores se corresponden con $125 = 100\mu s = 0.1ms = 1cm = 100\%$ y $31250 = 25000\mu s = 25ms = 400cm = 0\%$. Notar que, como en el telémetro, se encuentra invertida la relación de máximo y mínimo, el valor $125 = 1cm$ es el máximo, donde el sonar está viendo mucho o un objeto muy cercano, y el valor $31250 = 400cm$ es el mínimo, donde el telémetro está viendo poco o un objeto muy lejano.

Una vez obtenido e interpretado el valor del sensor, se satura en el rango `[125:31250]` y se procede a normalizar este valor teniendo en cuenta que se encuentran invertidos el máximo y mínimo. Para esto simplemente realizamos una conversión lineal (regla de tres simple) entre el valor del sensor y los valores máximos y mínimos invertidos. Una vez normalizados estos valores, pueden ser entregados el módulo GUI en el rango `[0%:100%]`.

setEstadoActuadores(): Esta función contiene la desnormalización (a valores absolutos) de los datos de motores antes de ser enviados al robot.

La GUI envía a la RAL valores para los motores normalizados en el rango `[-100%:100%]`, por lo que esta función se encargará de convertir estos valores a valores absolutos para ser entregados al robot en su rango absoluto `[-30:30]`. De esta forma, simplemente se procede a realizar una conversión lineal (regla de tres simple) entre los valores normalizados y

desnomalizados, donde $-30 = -100\%$ y $30 = 100\%$. Una vez desnormalizados estos valores, pueden ser entregados al robot en el rango `[-30:30]`.

Una cuestión que se realiza durante el proceso es saturar los valores mínimos de los motores. Los motores del ExaBot no funcionan correctamente si se les asigna menos del 20% de su capacidad, por lo que es necesario manejar los valores en estos intervalos pequeños. Para esto, definimos la siguiente saturación de valores antes de desnormalizar:

```
(-10:10)   ⟶  0%
[10:20)    ⟶  20%
(-20:-10]  ⟶  -20%
```

Otra cuestión a tener en cuenta es que los motores del robot giran invertidos, por lo que para que ambos motores vayan en el mismo sentido la RAL debe enviar al robot un valor de motor como positivo (+) y otro negativo (−) según el sentido.

Nota: tener en cuenta al hacer los cálculos de normalización-desnormalización que deben manejarse los datos en valores `float` para no perder precisión ni entrar en casos en los que devuelva *cero* por truncamiento a `int`.

5.9.3 Conexión por UDP

La conexión con el robot se establece mediante el protocolo UDP. Básicamente la implementación consiste en dos procesos (threads) que corren en paralelo: `udp_receive` y `udp_send`, que comparten la memoria para intercomunicarse mediante un proceso central.

Un detalle importante de la implementación es que fue necesario configurar como desacoplados del módulo Core a los procesos `udp_receive` y `udp_send`. El Core es el que crea estos procesos y, al terminar el Core, esto también terminaba sus procesos hijos (`udp_receive` y `udp_send`) y perdíamos la comunicación con el robot. Para desacoplar (*desatachear*) estos procesos del Core se utilizó la función `setsid()`.

5.9.3.1 Configuración IP

Se estableció la siguiente configuración de direcciones IPs para la conexión UDP:

```
Cable Ethernet IP ExaBot  ⟶  0xC0A80032  ⟶  192.168.0.50
Cable Ethernet IP PC      ⟶  0xC0A80033  ⟶  192.168.0.51  | 255.255.255.0 | 192.168.0.1

WiFi IP ExaBot            ⟶  0xC0A80132  ⟶  192.168.1.50
WiFi IP PC                ⟶  0xC0A801FE  ⟶  192.168.1.254 | 255.255.255.0 | 192.168.1.1
```

Definiciones IPs: Las IPs, tanto en el ExaBot como en la RAL-Exabot, están definidas en tiempo de compilación. Por lo que los ejecutables y librerías dinámicas tienen definido de antemano la IP (cable o WiFi) a la que se conectarán. Para la RAL, estas definiciones se encuetran en `exabotRAL.h` en las constantes `IP_EXA_CABLE` y `IP_EXA_WIFI`. Para la aplicación del robot, estas definiciones se encuetran en `udp_send.c`.

Notar que si las direcciones IPs no coinciden, no sólo en los códigos de las aplicaciones que se ejecutan, sino también físicamente en las placas de red del robot y la PC host, la conexión no podrá ser establecida.

La *submask* y *gateway* no son neesarios, pero conviene definirlos.

5.9.4 Software ExaBot para conexión

El robot cuenta con software específico para la conexión UDP, vía cable ethernet y WiFi, para enviar y recibir los comandos necesarios. Para que la GUI pueda comunicarse con

el robot, es necesario previamente que la aplicación de conexión UDP esté corriendo en el robot.

5.9.4.1 Aplicación UDP

La aplicación de conexión del robot se encuentra en el ExaBot en su PC embebida en `/pc104/1004_codigo_completo`. Se ejecuta desde esa ubicación con `./test_threads`. Para ampliar los distintos comandos de esta aplicación ver **??**.

AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!
AGREGAR SECCION SOFT ADICIONAL MANUAL DE COMANDOS DEL EXA !!!

En esta aplicación, al igual que en RAL-ExaBot, la IP de conexión a la PC remota (GUI) está definida en tiempo de compilación, por lo que el ejecutable se encuentra definido de forma fija para un tipo de IP y conexión. Para facilitar esto, ya se encuentran en `/usr/bin/` compiladas las dos versiones: `gui_test_threads` (cable ethernet) y `gui_test_threads_254` (WiFi) según la tabla de IPs anterior.

El ejecutable `gui_test_threads` conectará entre `192.168.0.50` (robot) y `192.168.0.51` (PC host).

El ejecutable `gui_test_threads_254` conectará entre `192.168.1.50` (robot) y `192.168.1.254` (PC host).

Lo más cómodo, si es que se utilizará intensivamente el robot junto con la GUI, es que esta aplicación de conexión se ejecute automáticamente en el boot del robot. Para esto, ver 5.9.4.2

Compilación: Si fuese necesario recompilar esta aplicación, el código fuente se encuentra en `/pc104/1004_codigo_completo` en la PC104. Es necesario conectarse por `telnet 192.168.0.50`, se hace `rm test_threads` y `make test_threads`, y con `./test_threads` se ejecuta para que espere comandos.

5.9.4.2 Boot automático en ExaBot:

poner cómo hacer para que bootee automático la aplicación!!
poner cómo hacer para que bootee automático la aplicación!!
poner cómo hacer para que bootee automático la aplicación!!
poner cómo hacer para que bootee automático la aplicación!!
poner cómo hacer para que bootee automático la aplicación!!

ADEMÁS hay que cargar *loadUSBModules.sh* y *loadUSB.sh* para que anden el PenWiFiUSB...

-> en EXABOT: pusimos el `gui_test_threads_254` en el booteo !!!!
-> `"PC104"/usr/bin/gui_test_threads_254`
-> `"PC104"/usr/bin/loadGuiWifi.sh`
-> `"PC104"/etc/rc.local`
-> `"PC104"/etc/rc.d/rc3.d/S99loadGuiWifi`

5.9.5 Conexión WiFi desde PC host

Por cuestiones de seguridad y estandarización, definimos que la conexión entre el robot y la PC host se realiza a través de un router LinkSys. En el router se encuentra configurado para que acepte por WiFi únicamente 2 MacAddress, la del pendriveWifi del robot y el pendriveWifi de la PC host.

Para la PC host utilizamos el pendriveWifi de marca *IOgear* (color blanco) macaddress `00:02:72:6A:E0:21`. Para el robot utilizamos el pendriveWifi de marca *Eusso* (color azul con antena) macaddress `00:02:72:69:28:B0`.

Además, configuramos el router para que asigne dinámicamente (DHCP) al pendriveWifi de la PC host la IP `192.168.1.254`. De esta forma, el robot tiene configurada la IP `192.168.1.50` que el router acepta por su macaddress y la PC host se conecta normalmente a través del sistema operativo, como a un router wireless común, con el pendriveWifi *IOgear* obteniendo por DHCP la IP `192.168.1.254`. El router wireless tiene definido como Network Name (SSID): *exabot*.

Utilizamos específicamente estos pendriveWiFi por el *chipset zd1211* que poseen, ya que según ARM (fabricante PC104) el TS-Kernel (Kernel de la PC104) está preparado para soportar este chipset... Si quiciéramos usar otro chipset, sería necesario conseguir drivers y recompilar el kernel del robot...

5.9.5.1 Configuración Router Wireless

Utilizamos un router LinkSys Wi-Fi WRT54G como router ExaBot. La configuración es la siguiente:

- http://192.168.1.1/, user: admin, pass: iogear.
- Setup > Basic Setup:
  - Automatic Configuration - DHCP
  - Local IP Address: 192.168.1.1
  - Subnet Mask: 255.255.255.0
  - DHCP Server: Enabled
  - Starting IP Address: 192.168.1.254
  - Maximum Number of DHCP Users: 1
- Wireless > Basic Wireless Settings:
  - Wireless Configuration: Manual
  - Wireless Network Mode: Mixed
  - Wireless Network Name (SSID): exabot
  - Wireless Channel: 6
  - Wireless SSID Broadcast: Enabled

- Wireless > Wireless Security:
  - Security Mode: Disabled
- Wireless > Wireless MAC Filter:
  - Wireless MAC Filter: Enabled (Permit only)
    - `00:02:72:6A:E0:21` - Pen WiFi IOGear (Blanco)
    - `00:02:72:69:28:B0` - Pen WiFi Eusso (Azul con antena)
    - `00:1B:77:86:2F:B6` - Notebook Matias
    - `D8:5D:4C:89:F7:89` - Pen WiFi TP-LINK TL-WN422G (Blanco nuevos con antena)
    - `D8:5D:4C:89:8F:4B` - Pen WiFi TP-LINK TL-WN422G (Blanco nuevos con antena)
    - `D8:5D:4C:89:97:FA` - Pen WiFi TP-LINK TL-WN422G (Blanco nuevos con antena)

Importante: Esto hace que sólo asigne la IP `192.168.1.254` y sólo al “Pen WiFi IOGear (Blanco)”. O sea que, cualquier PC que se conecte con ese "Pen WiFi IOGear (Blanco)" va a tener la IP `192.168.1.254`. El robot por WiFi siempre debe comunicarse a esa IP !!!

Nota: Es importante chequear dos cosas:

1. Que la IP del ExaBot esté configurada correctamente según el caso.
2. Que esté corriendo la aplicación correcta (Cable o Wifi) de conexión en el robot.

de lo contrario la conexión no podrá ser establecida.

5.10 RAL SimuladorExaBot

5.10.1 SimuladorExaBot

5.10.2 Implementación

5.10.2.1 Normalización de Sensores y Actuadores

FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!
FALTA VER ESTO!!! VER EN CODIGO DE MATI!!!!

Chapter 6

GUI: Graphical User Interface

6.1 Introducción

The GUI module is in charge of interfacing with the user. First, the user selects a robot or simulator to work with, and which sensors and actuators of the robot is going to use for this particular behaviour. The GUI allows the user to drag and drop the different objects (sensors, actuators, functions) to a work canvas, and then connect them using the mouse. Different functions may be selected from a menu, dragged to the canvas, and then configured with a pop-up configuration window.

Fig. 3 shows a screenshot of the GUI and Fig. 4 an example of the pop-up configuration window. Once the behaviour is finished, the user can select a robot to execute it on. The created behaviour and the minimum needed sensor and actuator configura- tion for its execution are stored in a file (the behaviour-file), that will be read by the CORE. The execution of the behaviour may be started and paused at any moment from the GUI. The GUI also provides general operations to open and save files.

Este módulo se encarga de la interfaz con el usuario y su función principal es la de permitir la programación gráfica del comportamiento del robot. El módulo GUI cuenta con las siguientes funcionalidades:

- Permitir en modo gráfico diseñar el modelo de Braitenberg mediante la interconexión de sensores con actuadores. Cada una de estas conexiones debe implementar funciones matemáticas parametrizables. De esta forma se define un *grafo de ejecución* que representa el comportamiento a realizar, donde los nodos son sensores, actuadores o funciones matemáticas. En la Figura 6.1 se muestra esta idea.

Figure 6.1: Un ejemplo de grafo de ejecución

- Permitir en modo gráfico diseñar una arquitectura de subsumisión para coordinar los distintos comportamientos.
- Realizar chequeos para validar los comportamientos diseñados y su coordinación.
- Guardar en un archivo el comportamiento diseñado y la configuración de sensores y actuadores requerida en un robot para poder llevar adelante ese comportamiento. Este archivo será leído y ejecutado por el Core.
- Ejecutar la aplicación, indicándole al CORE cuándo iniciar y finalizar la ejecución del comportamiento.
- Guardar y cargar configuraciones de distintos robots (sensores y actuadores).
- Realizar un replay de la experiencia, utilizando para ello un archivo generado por el Core durante la ejecución donde se almacena el estado de los sensores y actuadores en cada momento (archivo de LOGs).

- Replay (Debug). Leer el *LOG* para cuando se esté debuggeando e ir mostrando en la pantalla el estado de la máquina de estados, encendiendo con colores las cosas que se van activando para saber qué es lo que pasó...
- WebCam. De alguna forma, cuando el *RAL* es un robot real, se debería poder seleccionar que una WebCam grabe lo que sucede. Así sería un "debugging" para un robot real. Esto respetaría la filosofía de que no es posible debuggear como estamos acostumbrados, las cosas en un robot no funcionan así. Entonces, lo grabo y lo reproduzco en camara lenta...

FALTA PONER IMAGENES DE EUROBOT DE GUI!!!
FALTA PONER IMAGENES DE EUROBOT DE GUI!!!
FALTA PONER IMAGENES DE EUROBOT DE GUI!!!
FALTA PONER IMAGENES DE EUROBOT DE GUI!!!
FALTA PONER IMAGENES DE EUROBOT DE GUI!!!

Cómo es el uso y configuración de las funciones??? Por ejemplo, al agregar función exitatorio o inhibitoria en realidad de fondo es una paramétrica donde por defecto tiene los valores `(0,0):(1024,100)` ó `(0,0):(1024,-100)` según corresponda, estos varlores por defecto se definen en el archivo de configuración de la GUI, ver Capítulo 7.

6.2 Implementación

The GUI is implemented in Java, since it is a good language for graphical in- terfaces and its portable to several operating systems, only requiring the installa- tion of the JVM (Java Virtul Machine). The behaviour-file is an XML (Extensible Markup Language) file [1], making it very simple to add new robots, sensor types, functions and other features we might add to ERBPI.

El módulo GUI está implementado en `Java`. Elegimos este lenguaje por la capacidad de portabilidad y la no necesidad de recompilar para distintos Sistemas Operativos. El único requerimiento en la PC para ejecutar la GUI es tener instalado el JVM (Java Virtual Machine).

Para desarrollar la interfaz gráfica, usamos la *Swing API* (JFC/Swing). Ver `http://java.sun.com/docs/books/tutorial/uiswing/index.html` y `http://en.wikipedia.org/wiki/Swing_(Java)`

ARREGLAR ESTO DE CÓMO ESTÁ ORGANIZADO EL CODIGO!!!
ARREGLAR ESTO DE CÓMO ESTÁ ORGANIZADO EL CODIGO!!!
ARREGLAR ESTO DE CÓMO ESTÁ ORGANIZADO EL CODIGO!!!
Está organizado en tres paquetes:

- model: aca esta toda la parte "funcional". Por ejemplo, la clase Program tiene el programa con sus cajas y conexiones y la clase Robot tiene la descripción de cada robot.
- gui: todo lo que tiene que ver con la interacción con el usuario (paneles, cajas, dibujos, intereacción con el mouse, etc).
- model.persist: carga y graba de archivos xml.
- utils: métodos que facilitan algunas tareas.
- thirdparty: librerías que bajé programadas por otras personas.

La conexión entre el modelo y la gui se da por el método *publish-suscribe*: hay definidas interfaces de *listener*, y las clases pueden suscribirse a diferentes acciones. Por ejemplo, la clase *JConnectionsPanel*, que dibuja las conexiones, se suscribe al programa para que le avise cuando se genera una nueva conexión. También, por ejemplo, el panel con el esquema del robot se suscribe a la clase Robot para que le avise cuando algun sensor entra en "foco" y lo pinta.

La parte de gui es la más enquilombada, pero no pude hacerlo más fácil.

6.2.1 Configuraciones

Como ya dijimos, una de las particularidades de la aplicación es su parametrizabilidad y extensibilidad. Por esto mismo, todas las configuraciones que pueden realizarse a la GUI, se establecen, agregan y/o modifican a través de archivos de configuración sin la necesidad de modificar la aplicación. Lo que hace sencillo y rápido cambiar las características de la aplicación y los robots y simuladores a utilizar por la misma.

6.2.1.1 Configuración de la GUI

Utilizamos XML [1] para su implementación. Esto nos da la posibilidad de modificar y extender las caractersticas de la aplicación en cualquier momento. Por ejemplo, qué tipo de sensores y actuadores será capaz de manejar la aplicación, como *proximidad*, *luz*, *telemetro*, *sonar*, *linea*, *contacto*, *rueda*, y cuáles son las imágenes que estos utilizaran para visualizarse en la GUI. Cuáles son las herramientas de que disponemos para realizar las conexiones entre los componentes, como las funciones *inhibitoria*, *exitatoria*, *parametrica* y *constante*, si éstas aceptan otros componentes como entrada, sus parámetros por defecto y establecer las imágenes que estos utilizaran para visualizarse en la GUI.

Para más información sobre la configuración de la GUI ver Capítulo 7.

6.2.1.2 Configuración de los Robots

Utilizamos XML [1] para su implementación. Esto nos da la posibilidad de modificar y extender los robots y sus caractersticas que la aplicación podrá manejar a través del menú de selección de robot en la GUI. Cada robot tiene su propio archivo de configuración alojado en *gui/extension/robot*, por ejemplo *khepera.xml*, *yaks.xml*, *exabot.xml* y *exabot-player.xml*. Cada archivo de configuración de robot nos permite definir cuál es la RAL que deberá utilizar para comunicarse correctamente con el robot, qué sensores y de qué tipo posee el robot, cuáles de éstos aparecen por defecto en la aplicación y también la definición gráfica para la GUI del robot y sus sensores que permitirá poder visualizar en la aplicación a qué sensores y actuadores nos referimos con tan sólo pasar el mouse sobre cada componentes en el escritorio de trabajo.

Para más información sobre la configuración de los robots ver Capítulo 7.

6.3 Ejecución

6.3.1 Linux 32 bits

La GUI se ejecuta de la siguiente forma: *java extension.ExtensionApp*.

6.3.1.1 Script

Para facilitar esta ejecución, se incluye un archivo script *ERBPI/bin/gui/gui_ejecutar.sh*.

6.3.1.2 BUGs

Hay un bug en la ejecución, ver 11.2.1

6.3.2 Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

6.4 Compilación

6.4.1 Linux y Windows 32 bits

El código fuente de la GUI cuenta con los siguientes archivos:

- ./gui:
  - .classpath
  - .project
  - gui_ejecutar.sh
  - gui_instalar.sh
- ./gui/extension:
  - config.xml
  - ExtensionApp.java
- ./gui/extension/gui:
  - BoxColumnLayout.java
  - ComponentDragger.java
  - JBox.java
  - JBoxPanel.java
  - JBoxTemplate.java
  - JConnectionsPanel.java
  - JInlineDialog.java
  - JParametrosCajaEnergia.java
  - JParametrosCaja.java
  - JProgramPanel.java
  - JRoboticaFrame.java
  - JRobotPanel.java
  - PopupMenuMouseAdapter.java
- ./gui/extension/images:
  - activar_no.png

  - activar_si.png
  - act_rueda.png
  - conexion_cola.png
  - conexion_punta.png
  - f_energia.png
  - f_exitatoria.png
  - f_inhibitoria.png
  - f_parametrica.png
  - menu_abrir.png
  - menu_ejecutar.png
  - menu_guardar.png
  - menu_nuevo.png
  - menu_pausa.png
  - menu_salir.png
  - seleccionar.png
  - sen_contacto.png
  - sen_linea.png
  - sen_luz.png
  - sen_proximidad.png
  - sen_sonar.png
  - sen_telemetro.png
- ./gui/extension/model:
  - ActuatorBox.java
  - ActuatorType.java
  - Box.java
  - BoxListener.java
  - ConnectionMaker.java
  - ConnectionMakerListener.java
  - Diagram.java
  - FunctionBox.java
  - FunctionTemplate.java
  - GlobalConfig.java
  - ImageMapFeature.java
  - ImageMap.java
  - Panel.java
  - Program.java
  - ProgramListener.java
  - Robot.java

  - RobotListener.java
  - SensorBox.java
  - SensorType.java

- ./gui/extension/model/persist:
  - GlobalConfigXml.java
  - ProgramXml.java
  - RobotXml.java

- ./gui/extension/robots:
  - exabot.xml
  - khepera.xml
  - robot_exabot.png
  - robot_khepera.png
  - robot_yaks.png
  - yaks.xml

- ./gui/extension/utils:
  - FileUtils.java
  - IconBank.java
  - XmlUtils.java

- ./gui/thirdparty/dragnghost:
  - AbstractGhostDropManager.java
  - DragnGhostDemo.java
  - DragnGhostDemo.jnlp
  - GhostComponentAdapter.java
  - GhostDropAdapter.java
  - GhostDropEvent.java
  - GhostDropListener.java
  - GhostDropManagerDemo.java
  - GhostGlassPane.java
  - GhostMotionAdapter.java
  - GhostPictureAdapter.java
  - GlassPaneExtension.java
  - HeaderPanel.java
  - UIHelper.java

Para la compilación y debugging de este código, se cuenta con un *Eclipse Proyect* cuyas definiciones se ecuentran en los archivos *gui/.project* y *gui/.classpath*.

6.5 Instalación

6.5.1 Linux 32 bits

Para la instalación se ejecuta el script *ERBPI/src/gui/gui_instalar.sh* que se encarga de copiar los binarios en *ERBPI/bin/gui*.

Nota: Es necesario que la GUI se encuentre compilada con anterioridad. Para compilación de GUI ver punto 6.4.

6.5.2 Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

6.6 Compiladores

6.6.1 Java Linux 32 bits

Utilizamos el siguiente paquete: OpenJDK 6 (openjdk-6-jdk).

Con el paquete Open Source Java Development Kit obtenemos compilador e intérprete para Java Standard Edition.

6.6.2 Java Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

Chapter 7

XML: Configuración y Comportamiento

7.1 Introducción

El archivo *XML* servirá, por un lado, para la definición de datos que la *GUI* establecerá para que el *Core* ejecute. Por otro lado, el *XML* contendrá también información propia de la *GUI*.

LO ANTERIOR NO ES DEL TODO CIERTO, ARREGLAR!!!!
LO ANTERIOR NO ES DEL TODO CIERTO, ARREGLAR!!!!
LO ANTERIOR NO ES DEL TODO CIERTO, ARREGLAR!!!!

El archivo *XML* contendrá varias cosas:

- Los datos necesarios para que el *Core* pueda realizar la ejecución.
- Los datos necesarios que la GUI requerirá para poder funcionar, como las especificaciones gráficas, objetos, ubicación de los mismos, etc; y todas las opciones sobre los proyectos realizados...
- ¿algo más?

De esta forma, en principio el *XML* podría tener en secciones separadas los datos para el *Core* y para la *GUI*. Tal vez, no necesariamente estén completamente separados. De modo que, por ejemplo, el *Core* deberá buscar en el *XML* sólo los datos necesarios para lograr la ejecución e ignorar el resto de los datos innecesarios...

7.2 Core Implementación

ESTO CAMBIO POR ALGO MÁS COMPLEJO QUE HIZO THOMAS, ARREGLAR!!!!
ESTO CAMBIO POR ALGO MÁS COMPLEJO QUE HIZO THOMAS, ARREGLAR!!!!
ESTO CAMBIO POR ALGO MÁS COMPLEJO QUE HIZO THOMAS, ARREGLAR!!!!

7.2.1 Datos para la Ejecución del Core

Básicamente, el *Core* busca en la "estructura de arbol" del *XML* el elemento raíz de nombre:

```
<conducta> ... </conducta>
```

Cualquier otro elemento distinto de $<$ *conducta* $>$ será ignorado.

Importante: El elemento < *conducta* > debe ser el primero en orden de definición dentro del *XML* ya que el *Core* parsea al *XML* utilizando la API que implementa el estandar DOM [2]. Cualquier otro elemento posterior es ignorado.

Entonces, la definición de los datos de la conducta a ejecutarse en el *Core* dentro del *XML* constara de 4 cosas:

```
<conducta>
    <sensores> ... </sensores>
    <timers> ... </timers>
    <contadores> ... </contadores>
    <comportamiento> ... </comportamiento>
</conducta>
```

7.2.1.1 Sensores

Por un lado esta la definición de los sensores existentes y su identificación (*id*). Por ahora, el *id* indicará todo lo referido al sensor, es decir, su tipo (sonar, telémetro, encoder, random) y su ubicación relativa al robot en ángulos (de 0^o a 360^o), por ejemplo:

```
<sensor id='sonar.0'/>
<sensor id='telemetro.20'/>
<sensor id='telemetro.340'/>
<sensor id='encoder.motor.izquierda'/>
<sensor id='encoder.motor.derecha'/>
<sensor id='sonar.180'/>
<sensor id='random'/>
```

Vimos de agregar un tipo de sensor *random*, que no sería un sensor real en el hardware, sino un sensor simulado en software para poder agregar "aleatoriedad"...

7.2.1.2 Timers

Los timers son elementos globales, que representan relojes abstractos que son simulados en el software del core, y son exclusiavamente elementos de este. El robot no tiene ningun conocimiento de ellos. Sirven como condiciones para disparar transiciones entre distintos comportamientos, y únicamente pueden ser reseteados como actualizaciones durante éstas.

Estos relojes llevan la cuenta del tiempo en segundos, y únicamente pueden ser reseteados o consultado su valor.

Comienzan en 0 al comenzar a ejecutarse el core y este se ocupa de actualizarlos automaticamente.

Su definición únicamente exige un string de identificación (*id*).

Ejemplo de uso:

```
<timers>
    <timer id="timer.1"></timer>
    <timer id="timer.2"></timer>
</timers>
```

7.2.1.3 Contadores

Los contadores son otro tipo de elemento global, que como los timers, son simulados en el software del core e independientes del robot, y sirven igualmente para usar como condiciones para transiciones entre distintos comportamientos.

Los contadores son variables globales que almacenan un número. Comienzan en 0 al comenzar a ejecutarse el core, y pueden ser reseteados, incrtementados (+1) o decrementados (-1) únicamente durante las actualizaciones de una transición.

Su definición únicamente exige un string de identificación (*id*).

Ejemplo de uso:

```
<contadores>
    <contador id="contador.1"></contador>
    <contador id="contador.2"></contador>
</contadores>
```

7.2.1.4 Comportamiento

Los comportamientos constituyen el segundo y último nivel en una jerarquía de subsumisión de la conducta del robot. La conducta puede alternar entre distintos comportamientos mediante transiciones que se disparan bajo ciertas condiciones, que son revisadas en cada ciclo del *core*.

Un comportamiento está constituído por cajas (LLAMARLAS FUNCIONES, O ALGO MAS FELIZ EN ALGUN MOMENTO), actuadores y transiciones.

Su definición además requiere un string de identificación (*id*).

Ejemplo de uso:

```
<comportamiento id="comportamiento_adelante">
    <cajas> ... </cajas>
    <actuadores> ... </actuadores>
    <transiciones> ... </transiciones>
</comportamiento>
```

Sus elementos se detallan a continuación.

7.2.1.5 Cajas

Definen las *cajas* que representan las funciones que modifican los valores entre los sensores y actuadores. Estas *cajas* deben tener definidas sus *entradas* (sensores y otras cajas) y la función que se ejecuta sobre los respectivos valors de éstas. Por ahora sólo existe la "función partida" en tres tramos (constante + lineal + constante) cuya definición requiere dos puntos en el plano $(x_1; y_1)$ y (x_2, y_2). También es requerido un string de identificación (*id*) de la funcion.

Internamente, la salida de la *caja* será el resultado de aplicar la *función*, definida por los puntos $(x_1; y_1)$ y (x_2, y_2), a la sumatoria de los valores de todas sus entradas.

Ejemplo de uso:

```
<cajas>
    <caja id='caja1'>
        <entradas>
            <entrada id='sonar.0'/>
            <entrada id='telemetro.340'/>
            <entrada id='random'/>
        </entradas>
        <puntos>
            <punto x='100' y='0'/>
            <punto x='150' y='255'/>
        </puntos>
```

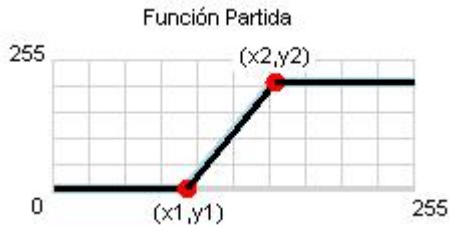

Figure 7.1: Aca vemos graficada la función partida en función de sus puntos.

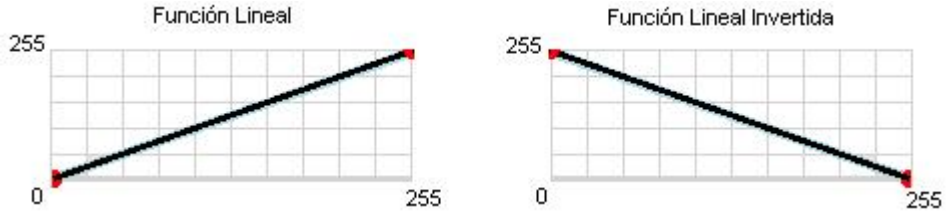

Figure 7.2: Las funciones *lineal* y *lineal invertida* podrían representarse usando funciones partidas con los puntos $\{(0;0),(255;255)\}$ y $\{(255;255),(0;0)\}$

```
    </caja>
</cajas>
```

SACAR LOS TITULOS DE LOS GRAFICOS EN LAS IMAGENES Y PONERLES UN TITLE A LAS FIGURAS.

SERIA FANTASTICO INCORPORAR Y EXPLICAR TAMBIEN UNA FUNCION CONSTANTE, LA QUE LLAMAMOS BOLA DE ENERGIA

7.2.1.6 Actuadores

Define los actuadores. Para cada uno, hace falta un string de identificación (*id*) así como los *ids* de los elementos cuyos valores usa como entradas. Internamente, la salida del *actuador* será la sumatoria de todas sus entradas.

Ejemplo de uso:

```
<actuadores>
    <actuador id='motor.izquierda'>
        <entradas>
            <entrada id='caja1'/>
            <entrada id='caja2'/>
        </entradas>
    </actuador>
    <actuador id='motor.derecha'>
        <entradas>
            <entrada id='caja2'/>
            <entrada id='telemetro.1'/>
        </entradas>
    </actuador>
</actuadores>
```

7.2.1.7 Transiciones

Define las transiciones que existen para cambiar de éste comportamiento a otro. Las mismas, además de un string de identificación (*id*), requieren la definición de las condiciones (necesarias y suficientes) para su activación, así como de las actualizaciones que se ejecutan sobre elementos globales, como timers y contadores, al activarse la misma.

Ejemplo de uso:

```
<transiciones>
    <transicion id="transicion.1" id_destino="comportamiento_atras">
        <condiciones> ... </condiciones>
        <actualizaciones> ... </actualizaciones>
    </transicion>
</transiciones>
```

Sus elementos se detallan a continuación.

7.2.1.8 Condiciones

Las condiciones son expresiones booleanas necesarias y suficientes que se deben cumplir para que se dispare la transición a la cuál refieren. La expresión requiere de un elemento, de un tipo de comparación y de un umbral. El valor de verdad de la condición se obtiene comparando el valor del elemento de la forma especificada contra el valor del umbral. Los valores que puede tomar el campo de tipo de comparación son *igual*, *menor*, *menor_igual*, *mayor* y *mayor_igual*.

Ejemplo de uso:

```
<condiciones>
    <condicion id_elemento='proximidad.350' comparacion='mayor' umbral='1' />
    <condicion id_elemento='proximidad.10' comparacion='menor_igual' umbral='90' />
</condiciones>
```

7.2.1.9 Actualizaciones

Define las actualizaciones que se disparan al activarse la transición a la que refieren. Las actualizaciones pueden ser sobre *Timers* o *Contadores* unicamente, siendo esto especificado en una propiedad de tipo que únicamente toma los valores *timer* o *contador*, acompañado del string identificatorio (*id*) del *Timer* o *Contador* correspondiente. En el caso de un *Timer*, Se da por sentado que se lo quiere resetear, la cuál es la única accion disponible sobre este elemento. En el caso de un *Contador*, se requiere además el tipo de acción a ejecutarse, siendo los únicos valores legítimos *resetear*, *incrementar* o *decrementar*. Cada actualizació requiere por supuesto un string identificatorio (*id*).

Ejemplo de uso:

```
<actualizaciones>
    <actualizacion tipo="timer" id_timer='timer.1'/>
    <actualizacion tipo="contador" id_contador='contador.1' accion="incrementar" />
</actualizaciones>
```

7.2.1.10 Importante

Las etiquetas y atributos definidos anteriormente deben ser estrictamente definidos de esa forma en el *XML* (en minúsculas). **Cualquier otra etiqueta o atributo distinto será ignorado.**

7.3 GUI Implementación

DOCUMENTAR CADA PARAMETRO, MOSTRAR TAMBIÉN LOS PNGs (SENSORES Y ROBOTS)...

Los labels más relevantes son (COMENTAR CADA UNO!!):

gui/extension/config.xml:

```
        <tipoSensores>
                proximidad, luz, telemetro, sonar, linea, contacto
                y establecer sus respectivas imagenes
        <tipoActuadores>
                rueda
                y establecer sus respectivas imagenes
        <herramientas>
                inhibitoria, exitatoria, parametrica, energia
                aceptaentradas
                y establecer sus respectivas imagenes y configuraciones por defecto...
```

gui/extension/robots:

```
        <robot id="exabot" nombre="ExaBot">
                <imagen id='esquema' href='robots/robot_exabot.png' />
                <ral href='exabotRAL.so'/>
        <sensores>
                    <sensor id='telemetro.315' tipo='telemetro' default='true'>
                                <mapaimagen imagen='esquema'>
                                        <linea x0='0.36' y0='0.35' x1='0.41' y1='0.30' width='5'
                                </mapaimagen>
                    <sensor id='sonar.0' tipo='sonar' default='true'>
        <actuadores>
                <actuador id='motor.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=
                        </mapaimagen>
                        <ubicacion id='izquierda'/>
```

Como en el caso anterior, utilizamos labels que denotan las distintas caractersticas de la aplicacion. Un ejemplo sen- cillo de GUI setting file puede ser:

Para agregar un nuevo robot para trabajar con la aplicacion, solo deberemos programar el nuevo RAL correspondiente de ese robot e indicarle a la GUI en su archivo de configuracin la existencia del nuevo robot.

Hay un archivo xml que define cada robot, y uno con configuración general (*config.xml*). Desde ahí se puede cambiar las cajas que aparecen en las herramientas, los dibujos, etc. Los dibujos están todos en la carpeta images.

7.3.1 Datos para la GUI

Cómo es el uso y configuración de las funciones??? Por ejemplo, al agregar función exitatorio o inhibitoria en realidad de fondo es una paramétrica donde por defecto tiene

los valores `(0,0):(1024,100)` ó `(0,0):(1024,-100)` según corresponda, estos varlores por defecto se definen en el archivo de configuración de la GUI.

Para esto se toca *src/gui/extension/config.xml*

Lo mismo para la función constante, al agregar una de éstas, tiene por defecto el valor 10, que es 10%. Para esto también se toca *src/gui/extension/config.xml*. OJO: el CORE trata como si fuera una función partida más a la función constante, por eso sus valores deben ser: `(0,10):(1024,10)`.

7.3.2 Ejemplo

Originalmente (no nos habíamos dado cuenta) la FUNCION PARAMTETRICA, no acpetaba entradas... Entonces, en "config.xml" cambiamos a `<esquemacaja id='parametrica' aceptaentradas='true'>`

Chapter 8

Log

8.1 Introducción

El archivo de *log* de cada ejecución se encarga de esribirlo el *Core*. Siempre sobreescribe el archivo especificado, o lo crea si no existe, es decir, sólo queda en el archivo el contenido de la última ejecución.

El *log* tiene la siguiente especificación:

1. **La primera línea.** Consiste de la secuencia, separada por comas, de los *ids* de la tabla de ejecución en el orden en que se encuentran en la misma.
2. **Siguientes líneas.** Son todas iguales. Consiste de varios valores, separados por comas, de la siguiente forma:
   - *TimeStamp*. Es el tiempo en *milisegundos* para cada línea relativo al comienzo, es decir, comienza en cero.
   - *Valor de los elementos*. En el mismo orden en que fueron detallados en la primera línea, si es una `caja` son los valores *entrada* y *salida* de la caja, y si es un *sensor* o un *actuador* es simplemente el valor de *salida*.

Por ejemplo, el siguiente archivo de *log* corresponde a 10 ejecuciones del *Core*:

```
sonar.0, sonar.1, sonar.2, actuador.0, caja.0, caja.1, actuador.1,
0, 6, 8, 7, 8, 22, 13, 8, -9, 20,
103, 3, 7, 6, 7, 17, 13, 7, -8, 19,
204, 8, 3, 2, 3, 14, 13, 3, -6, 15,
304, 7, 4, 10, 4, 15, 13, 4, -6, 23,
405, 7, 3, 7, 3, 13, 13, 3, -6, 20,
505, 10, 8, 9, 8, 26, 13, 8, -9, 22,
606, 4, 3, 1, 3, 10, 13, 3, -6, 14,
707, 3, 10, 6, 10, 23, 13, 10, -13, 19,
807, 8, 10, 9, 10, 28, 13, 10, -13, 22,
1009, 3, 2, 7, 2, 7, 8, 2, -6, 15,
```

PENDIENTE
REPLAY Y DEBUG...

8.2 Core Implementación

Chapter 9

Software adicional

9.1 Xerces XML Parser

FALTA PONER UNA INTRO!!!

9.1.1 Compilación

9.1.1.1 Linux 32 bits - Librerías Estáticas

El código fuente del Core se compila incluyendo las librerías estáticas del Xerces para que estén incluidas en el ejecutable del Core y no sea necesario transportalas. Para eso, es necesario obtener el código fuente de las librerías del Xerces y recompilarlas de forma estática antes de poder compilar el Core.

Para recompilar las librerías del Xerces de forma estática, los pasos son los siguientes:

1. Bajar el archivo `xerces-c-3.0.1.zip` del código fuente del Xerces de `http://apache.xmundo.com.ar/xerces/c/3/sources/xerces-c-3.0.1.zip`
2. Descomprimir el archivo `xerces-c-3.0.1.zip` en alguna carpeta, por ejemplo, en */home/.../workspace/xerces-c-3.0.1-static/*
3. En la carpeta */home/.../workspace/xerces-c-3.0.1-static/xerces-c-3.0.1/* ejecutar "*./configure –disable-shared –disable-network*" para que no compile las librerías dinámicas (*.so*), y sólo compile las estáticas (*.a*). La opción *–disable-network* podría obviarse.
4. En la carpeta */home/.../workspace/xerces-c-3.0.1-static/xerces-c-3.0.1/* ejecutar "*make*" para compilar.
5. En la carpeta */home/.../workspace/xerces-c-3.0.1-static/xerces-c-3.0.1/* ejecutar "*sudo make install*" para instalar las librerías estáticas en el sistema. Por defecto, las mismas se instalan en */usr/local/bin*, */usr/local/lib*, */usr/local/include*.

Por último, para compilar cualquier código fuente que incluya las librerías, es necesario indicarle al *linker* que incluya las librerías estáticas */usr/local/lib/libxerces-c.a* y *libpthread.a*. Si esto último se hace desde alguna IDE de programación en `C++` simplemente se agrega en las opciones del programa, *Sección Linker*, la inclusión de las librerías mencionadas. Si la compilación se realiza manualmente, los parámetros son los siguientes:

```
g++ -static -o nombreEjecutable Codigo.cpp -lxerces-c -lpthread
```

9.1.1.2 Windows 32 bits - Librerías Estáticas

Para recompilar las librerías del Xerces de forma estática en *Windows*, los pasos son los siguientes:

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

9.1.2 Compiladores

9.1.2.1 C++ Linux 32 bits

Utilizamos los siguientes compiladores para C++:

- gcc: GCC (GNU Compiler Collection) C compiler.
- g++: GCC (GNU Compiler Collection) C++ compiler.

Ver http://gcc.gnu.org/

9.1.2.2 C++ Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! Para que los comandos como *gcc*, *g++*, *make* (*mingw32-make*) anden en la consola de windows, es necesario modificar la variable de sistema `PATH` de Windows y agregar la ruta `C:/MinGW/bin`.

FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

9.2 YAKS

YAKS es un simulador de cóìdigo abierto, escrito en `C++`, de robots tipo `Khepera`, desarrollado por Johan Carlsson. Su nombre proviene del acrónimo *Yet Another Khepera Simulator*. Posee las siguientes características:

- Permite incluir en el entorno obstáculos circulares, paredes, luces y definir zonas.
- Permite definir y manipular un número ilimitado de robots.
- Permite separar el programa de control del simulador, ya que los robots pueden ser manejados a través de una conexión `TCP/IP`.
- Soporta una gran variedad de sensores: proximidad, luminosidad, energía, encoders de las ruedas, compás y sensor de tierra (para detección de zonas).

PONER UNA IMAGEN DEL YAKS!!!!!

9.2.1 Ejecución

9.2.1.1 Linux 32 bits

El YAKS debe ejecutarse manualmente cada vez que se quiera utilizar com RAL desde la GUI. El YAKS se ejecuta de la siguiente forma: *gsim yaks-params.opt*. Debido a que es necesario setear parámetros, tanto para el YAKS en *yaks-params.opt* como en el sistema operativo para la librería GTK, se proporciona un script que tiene en cuenta estos detalles.

Script Para facilitar esta ejecución, se incluye un archivo script *ERBPI/bin/yaks/yaks_ejecutar.sh*.

Una vez compilado el *YAKS*, podemos proceder a su ejecución. Antes se deberán realizar las siguientes tareas:

- Copiar el ejecutable *gsim* del *YAKS* (que en la compilación fue creado en *src/bin*) en la carpeta raíz de los fuentes *src*.
- Crear el archivo `yaks-params.opt` de parámetros para la ejecución del *YAKS* en la carpeta raíz del *YAKS*. Puede verse un ejemplo de este archivo en http://www.exa.unicen.edu.ar/catedras/irobotic/yaks-params.htm.
  Es necesario tener en cuenta que la mayoría de estos ejemplos se encuentran hechos para Windows, por lo tanto es necesario editar todas las rutas de archivos y carpetas de ".\" a "./" para que funcione correctamente en Linux. Por ejemplo, en el archivo de parámetros de ejemplo, es necesario cambiar "`WORLD_PATH .\worlds`" por "`WORLD_PATH ./worlds`".
- Agregar en el archivo `yaks-params.opt` de parámetros la línea `CAMERA_PATH ./cam` para que el *YAKS* sepa dónde buscar estos archivos.
- Ejecución: ejecutar el simulador *YAKS* como `./gsim yaks-params.opt` en la carpeta raíz del *YAKS*.

Errores en la Ejecución Es muy posible que al intentar ejecutar, la librería *GTK* arroje errores de ejecución como `Gdk-ERROR`.

En Linux, esto se debe a que la librería *GTK* necesita que los efectos visuales de pantalla del sistema operativo estén deshabilitados.

En Windows, HAY PROBLEMAS ??? FALTA COMPLETAR ESTO!!!

Solución: Para deshabilitar los efectos visuales, se ejecuta en la consola el comando:
`export XLIB_SKIP_ARGB_VISUALS=1`

9.2.1.2 Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

9.2.2 Compilación

COMPLETAR ESTO CON http://profesores.elo.utfsm.cl/~tarredondo/memorias/2005-memoria-fquiros.pdf QUE EXPLICA MUUY BIEN TODO EL YAKS...

El código fuente del YAKS se encuentra en http://www.exa.unicen.edu.ar/catedras/irobotic/YAKS.htm En particular, bajamos dos archivos:

- http://www.exa.unicen.edu.ar/catedras/irobotic/SOFTWARE/YAKS-src-update-2.zip: Código Fuente Completo para Linux.
- http://www.exa.unicen.edu.ar/catedras/irobotic/SOFTWARE/yaks-linux-patch.tar.gz: Patch de YAKS para que no tire errores al compilarlo en Unix.

Luego se descomprime el contenido de los dos archivos fuentes en alguna carpeta, por ejemplo `yaks/src`.

9.2.3 Linux 32 bits

- En la carpeta donde se encuentra el código fuente del *YAKS*, creamos dos carpetas *bin* y *lib*, que son necesarias para crear los archivos resultantes de la compilación.
- Instalar la librería *GTK*, que es una librería estandard de `C++` para el manejo de ventanas gráficas. Se puede instalar de la siguiente forma: `sudo apt-get install libgtk1.2-dev`
- Compilar: en la carpeta donde se encuentra el código fuente del *YAKS*, ejecutar `make`

9.2.3.1 Errores en la Compilación

Es muy posible que al intentar compilar, el compilador arroje errores relacionados con `iostream.h`, `cout` y `endl` en los archivos fuentes `ann.cpp`, `ann.h`, `ga.cpp` y `ga.h`.

En Linux, este problema se origina al enlazar con la librería *GTK*, ya que no es posible utilizar `std::cout` porque no "entiende" a qué pantalla o consola tiene que mandar el *stream*, puesto que la librería *GTK* maneja varias pantallas.

Solución: Es necesario editar los archivos fuentes `ann.h`, `ann.cpp` y `ga.h`, sacando de los mismos las líneas `#include<iostream.h>` y `std::cout` (con sus respectivos operadores <<). Si se deseara mantener la impresión por pantalla, se deberían cambiar los `std::cout` por la función `printf` de `C`.

9.2.3.2 Error de Librería GTK

Al parecer, la librería *libgtk1.2-dev* está desactualizada (muy vieja) y no está más disponible en los repositorios de Ubuntu. La opción que queda es instalar la *libgtk2.0-dev*. Pero con esta nueva versión, el YAKS da errores de compilación.

Solución 1: Agregar en la lista de fuentes del *apt* el lugar desde donde bajar la *libgtk1.2-dev*, así:

```
sudo gedit /etc/apt/sources.list
```

agregar al final la línea "*deb http://cz.archive.ubuntu.com/ubuntu hardy main universe*" y ejecutar:

```
sudo apt-get update
sudo apt-get install libgtk1.2-dev
```

ahora debería instalarse automáticamente.

Si no anda, es necesario bajar y compilar a mano la librería desde: http://packages.ubuntu.com/hardy/i386/libgtk1.2-dev/download.

Solución 2: Corregir los fuentes del YAKS para que compile con la *libgtk2.0-dev*. Parece complicado... ESTO ESTÁ PENDIENTE !!!!

Y HAY QUE HCERLO PORQUE EN LOS LABOS NO DAN MAS SOPORTE PARA *libgtk1.2-dev*, SÓLO VAN A TENER LA *libgtk2.0-dev* !!!! HABLAR COM Maximiliano Geier !!!

9.2.3.3 Script

Para facilitar la compilacion, se incluye un archivo script "yaks_compilar.sh". El mismo puede ejecutar simplemente con `./yaks_compilar.sh`.

9.2.4 Windows 32 bits

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

9.2.5 YAKS para Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

9.2.6 Instalación

9.2.6.1 Linux 32 bits

Para la instalación se ejecuta el script *ERBPI/src/yaks/yaks_instalar.sh*. Es necesario que el YAKS se encuentre compilado con anterioridad. Para compilación de YAKS ver punto 9.2.2.

9.2.6.2 Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!

9.2.7 Compiladores

9.2.7.1 C++ Linux 32 bits

Utilizamos los siguientes compiladores para C++:

- gcc: GCC (GNU Compiler Collection) C compiler.
- g++: GCC (GNU Compiler Collection) C++ compiler.

Ver http://gcc.gnu.org/

9.2.7.2 C++ Windows 32 bits

FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!!
FALTA COMPLETAR ESTO!!! Para que los comandos como *gcc*, *g++*, *make* (*mingw32-make*) anden en la consola de windows, es necesario modificar la variable de sistema `PATH` de Windows y agregar la ruta `C:/MinGW/bin`.

FALTA COMPLETAR ESTO!!! Ojo con esto, porque aunque simula el GNU-GCC, no necesariamente todas las librería incluidas andan, porque algunas son especificas de Linux, por ejemplo "sys/socket.h", que me parece que en Windows hay que cambiarla por "winsock.h'0'. HAY QUE VER BIEN ESTO y ANOTAR !!!!

FALTA COMPLETAR ESTO!!! Nota: cualquier cosa, probar también Cygwin 5.1.6 (GNU + Cygnus + Windows) que contiene de http://www.cygwin.com/. ojo con esto porque me parece que sí o sí necesita "cygwin1.dll" en la PC para que después pueda andar... Probar!!!

9.3 SimuladorExaBot

Chapter 10

Compilación, Instalación y Ejecución

10.1 Compilación

10.2 Instalación

Estructuralmente el software se divide en dos carpetas:

1. ***ERBPI/src:*** contiene todos los fuentes necesarios del software para su compilación.
2. ***ERBPI/bin:*** contiene todos los ejecutables, producto de la compilasión de los fuentes, necesarios para la ejecución completa del software.

Cada módulo y RALs contienen su fuentes y, por lo tanto, sus scripts necesarios para su compilacion e instalación. La instalación de cada módulo produce la estructura de binarios siguientes:

- ERBPI/bin:
  - /core
  - /gui
  - /yaks

Opcionalmente también se introduce en la carpeta *ERBPI/bin/comportamientos* que contiene archivos *XML* con comprtamientos estándard ya programados.

10.3 Ejecución

Chapter 11

Bugs

Estos son bugs conocidos, algunos arreglados y otro no...

11.1 Sin Arreglar

11.1.1 Khepera RAL

A veces el KHEPERA-RAL no arranca, tira `segmentation fault`... Parece ser un problema de que no puede abrir bien el puerto COM... Hay que darle hasta que deje de devolver `segmentation fault`...

11.1.2 Khepera RAL Torreta-Radio

Parece haber un problema con el *setMotors(motors)* en *setEstadoActuadores()* en el *RAL.CPP*. Si comentamos esto, la lectura de sensores a *CTR_FREC = 700000* anda bárbaro, si no, empieza a fallar...

El error típico parece ser que se pierden lecturas y devuelve todo cero. Una salida de log típica cuando está activado el *setMotors(motors)* de la función *setEstadoActuadores()* en *RAL.CPP* es:

```
timestamp, proximidad.320, proximidad.340, proximidad.350, proximidad.10, proximidad.20, proximidad.40, proximidad.170, proximidad.1
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
701, 1023, 1023, 141, 0, 0, 0, 0, 0, 2187, 10, 0, 3, 0, 0, 2187, -10, 10, -7,
1402, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
2103, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
2804, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
3505, 1023, 1023, 393, 0, 0, 0, 0, 0, 2439, 10, 0, 3, 0, 0, 2439, -10, 10, -7,
4206, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
4907, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
5607, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
6307, 1023, 1023, 603, 0, 0, 0, 0, 0, 2649, 10, 0, 3, 0, 0, 2649, -10, 10, -7,
7008, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
7710, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
8410, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
9111, 1023, 1023, 244, 0, 0, 0, 0, 0, 2290, 10, 0, 3, 0, 0, 2290, -10, 10, -7,
```

En cambio, si lo comentamos empieza a andar todo bien:

```
timestamp, proximidad.320, proximidad.340, proximidad.350, proximidad.10, proximidad.20, proximidad.40, proximidad.170, proximidad.1
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 3, 3,
702, 677, 1023, 0, 0, 0, 0, 0, 0, 1700, 10, 0, 3, 0, 0, 1700, -10, 10, -7,
1403, 1023, 1023, 292, 0, 0, 0, 0, 0, 2338, 10, 0, 3, 0, 0, 2338, -10, 10, -7,
2103, 1023, 1023, 136, 0, 0, 0, 0, 0, 2182, 10, 0, 3, 0, 0, 2182, -10, 10, -7,
2804, 1023, 1023, 170, 0, 0, 0, 0, 0, 2216, 10, 0, 3, 0, 0, 2216, -10, 10, -7,
3504, 1023, 1023, 102, 0, 0, 0, 0, 0, 2148, 10, 0, 3, 0, 0, 2148, -10, 10, -7,
4205, 1023, 1023, 174, 0, 0, 0, 0, 0, 2220, 10, 0, 3, 0, 0, 2220, -10, 10, -7,
4905, 1023, 1023, 330, 0, 0, 0, 0, 0, 2376, 10, 0, 3, 0, 0, 2376, -10, 10, -7,
5605, 1023, 1023, 261, 0, 0, 0, 0, 0, 2307, 10, 0, 3, 0, 0, 2307, -10, 10, -7,
6306, 1023, 1023, 180, 0, 0, 0, 0, 0, 2226, 10, 0, 3, 0, 0, 2226, -10, 10, -7,
7707, 1023, 1023, 144, 0, 0, 0, 0, 0, 2190, 10, 0, 3, 0, 0, 2190, -10, 10, -7,
8407, 1023, 1023, 245, 0, 0, 0, 0, 0, 2291, 10, 0, 3, 0, 0, 2291, -10, 10, -7,
9108, 1023, 1023, 125, 0, 0, 0, 0, 0, 2171, 10, 0, 3, 0, 0, 2171, -10, 10, -7,
9808, 845, 1023, 0, 0, 0, 0, 0, 0, 1868, 10, 0, 3, 0, 0, 1868, -10, 10, -7,
10508, 1023, 1023, 0, 0, 0, 0, 0, 0, 2046, 10, 0, 3, 0, 0, 2046, -10, 10, -7,
11209, 1023, 1023, 0, 0, 0, 0, 0, 0, 2046, 10, 0, 3, 0, 0, 2046, -10, 10, -7,
11909, 1023, 1023, 33, 0, 0, 0, 0, 0, 2079, 10, 0, 3, 0, 0, 2079, -10, 10, -7,
```

11.1.3 GUI Reseteo Workbench

Pregunta: ¿Se puede ejecutar el mismo comportamiento en dos robots diferentes? Por ejemplo, si sólo usas telemetros en el Exabot para evitar obstaculos, podes pasar el mismo comportamiento al Khepera? ¿Dónde está la normalizacion de los sensores, en el RAL?
Respuestas:

- La normalización de sensores está en RAL, entonces se podrían reutilizar los comportamientos para distintos robots porque todo pasa a ser relativo a 0-100%.
- Reutilizar comportamiento en disntintos robots: Objetivamente sí se puede. Pero en realidad, como no tuve tiempo de "tocarlo bien" en el código de Java, cuando cambiás en el menú de la GUI de selección de robot, te resetea el "escritorio de trabajo" (workbench) de la GUI... Es sólo un problemita menor de programación en la GUI... Sólo habría que tocar la GUI para que no resetee el workbench y vuele los sensores y actuadores (con sus conexiones) que no corresponden en función del nuevo robot...

Si programaste un comportamiento y cambiás el robot desde el menu de selección de robot, se reseteo todo!!! ARREGLAR ESTO!!!

Ver Bug Arreglado en 11.2.2

11.1.4 GUI lectura componentes - sensores

Hay 2 problemas: por cómo hice el código de la GUI en *JRobotPanel::JSensorsChooserDialog()* los IDs de los sensores deben ser `tipo.numero`. Si lo que está después del punto no es un número, no anda. Si lo que está antes del punto no es igual a unp de los `tipo`, no anda. Entonces, el formato debe ser:

```
<sensor id='telemetro.45' tipo='telemetro'>
```

11.1.5 RAL ExaBot

Falta hacer una función `inicializarRAL(lista sensores)` para que prenda sólo algunos sensores, los pasados por la lista. El problema es que por defecto la RAL prende todos los sensores del robot. Cuando esto no es necesario, tener todos los sensores prendidos consume mucha energía y las baterías del robot se gastan muy rápido.

Esto es principalmente para el Exabot, pero hay que dejarla establecida en el `.H` para todas las RALs...

11.1.6 GUI

Abrir la gui no maximizada. Abrir la pantalla de seleccion de robot. Maximizar la ventana. La pantalla queda de tamaño chiquito!!

11.2 Arreglados

11.2.1 GUI Ejecución

Tenemos un problema, después de darle ejecutar a la GUI (por ejemplo con el YAKS) el yaks se cerraba solo a los 10 segundos... El problema era que el applet (ventanita) de la GUI que indicaba que se estaba ejecutando (el CORE) no freezaba (detenía) la ejecución de la GUI, entonces inmediatamente lo que seguía a ejecutar era el `proc.destroy()` en la GUI. A esto se le sumaba que el `proc.destroy()` a veces no mataba bien el proceso y

quedaba pululando el CORE por ahí y empezaba a andar todo mal!!! imaginate dos COREs corriendo al mismo tiempo!!!...

No pudimos arreglar bien esto y por falta de tiempo lo EMPARCHAMOS horriblemente haciendo que el Java se freeze con una ventanita esperando el botón STOP y cuando se presione inmediatamente busque el IDPROC del CORE y mande `killall` via sistema para matar el CORE. ESTO ES HORRIBLE!!!! OJO!!! Esto sólo anda en LINUX!!!! Para que termine bien el CORE con SIGINT (Ctrl+C) lo arreglamos así:

```
kill -2 pid
ps -ef
kill -int pid
```

el pid tiene que ser el de

```
-> /lib/ld-linux.so.2 --library-path ../core ../core/core_exe /tmp/prg-4851298914852602110 core_log.txt yaks
```

y no el de

```
/bin/sh ../core/core.sh /tmp/prg-4851298914852602110 core_log.txt yaks
```

con

```
ps -ef | grep /lib/ld
```

obtengo listada sólo la linea del proceso que me interesa:

```
javier    8933  8932  0 17:52 ?        00:00:00 /lib/ld-linux.so.2 --library-path ../core ../core/core_exe /tmp/prg-
```

entonces la segunda "palabra" es el pid que me interesa...

Puede verse más en: http://www.devdaily.com/java/edu/pj/pj010016, http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4784574, http://ubuntuforums.org/showthread.php?t=995619, http://www.experts-exchange.com/Programming/Languages/Java/Q_22052834.html.

Al final, lo solucioné haciendo lo diguiente:

```
// obtengo el PID del "core_proc"
String[] params2 = new String[] { "/bin/bash", "-c",  "ps -ef | grep /lib/ld" };
Process core_proc_pid = Runtime.getRuntime().exec(params2);
BufferedReader core_proc_pid_br = new BufferedReader(new InputStreamReader(core_proc_pid.getInputStream()));
String core_proc_pid_br_line = core_proc_pid_br.readLine();
StringTokenizer core_proc_pid_br_line_st = new StringTokenizer( core_proc_pid_br_line );
String core_proc_PID;
core_proc_PID = core_proc_pid_br_line_st.nextToken();
core_proc_PID = core_proc_pid_br_line_st.nextToken();
System.out.println("-> el PID del core es: " + core_proc_PID);

// termina el core enviando el signal SIGINT // obtengo el PID del "core_proc"
String[] params3 = new String[] { "/bin/bash", "-c",  "kill -int " + core_proc_PID };
Process core_proc_SIGINT = Runtime.getRuntime().exec(params3);
System.out.println("-> hicimos 'kill -int " + core_proc_PID + "' y matamos el core...");
```

Ahora después de un tiempo, creo que el problema era que no estaba bien implementado el `SIGTERM` Y `SIGINT` en el CORE. Pero esto ya lo arreglamos. Habría que volver atrás esta modificación y ver que esté andando correctamente...

Este moco se encuentra en el archivo *JRoboticaFrame.java*.

11.2.2 GUI Menu Selección Robot

No andaba bien el menu de cambio de Robot. El problema era que trataba de buscar los XMLs de robots en *extension/robots* y tenía que buscarlos en *bin/extension/robots* ya que el programa ejecuta desde el directorio *gui* y no desde *gui/bin*...

Además, para que al cambiar el robot en el menu también cambie todo en la aplicación (se actualice) también hubo que tocar lo siguiente:

- `JRoboticaFrame::openProgram(File f)`: leemos del archivo cuál es el robot y actualizamos *programPanelHolder*, *robotPanel* y *robotNameLabel.setText(robot.getName())*.

- `JRoboticaFrame::actionPerformed(e.getActionCommand().startsWith("changeRobot:"))`: actualizamos *robot*, *robotNameLabel*, *programPanelHolder* y *robotPanel*.

OJO: Es decir que, si tenés un programa armado para un robot y tocás cambiar el robot, te borra todo y empieza de nuevo, porque no tiene sentido ya que los sensores se llaman distintos!!! ESTO ÚLTIMO NO ESTÁ BIEN!!! HAY QUE ARREGLARLO PARA QUE UN COMPORTAMIENTO PUEDA USARSE PARA DISTINTOS ROBOTS!!!! Ver Bug Sin Arreglar en 11.1.3...

11.2.3 GUI Configurar Parámetros de Funciones (Cajas)

Hubo que combinar con código Diego para que le pase a *cajaParametros* los 2 puntos para dibujarlos y que al presionar los boton aceptar devuelva a la GUI valor de los 2 puntos modificados...

¿Qué hicimos?
en JBox.java:

```
private JPopupMenu getPopupMenu() {
    ...
    item2.setActionCommand("setup");
    popup.add(item);
    if( box instanceof FunctionBox )
        popup.add(item2);
public void actionPerformed(ActionEvent e) {
    ...
    if( e.getActionCommand() == "setup" ) {
        Point A = new Point();
        Point B = new Point();
        A.x = ((FunctionBox) this.box).getX0();
        A.y = ((FunctionBox) this.box).getY0();
        B.x = ((FunctionBox) this.box).getX1();
        B.y = ((FunctionBox) this.box).getY1();
        JParametrosCaja setupBox = new JParametrosCaja( A,B, Program.getCurrentProgram(), this.box );
        setupBox.run();
    }
```

en FunctionBox.java:

```
public void setX0( int valor )...
public void setY0( int valor )...
public void setX1( int valor )...
public void setY1( int valor )...
```

en ProgramListener.java:

```
public interface ProgramListener {
    public void boxSet( Box box, Point A, Point B );
```

en Program.java:

```
public void setBox( Box box, Point A, Point B ) {
    for( ProgramListener listener: listeners )
        listener.boxSet(box,A,B);
```

en JConnectionsPanel.java:

```
public void boxSet( Box box, Point A, Point B ){}
```

en JBoxPanel.java:

```
public void boxSet( Box box, Point A, Point B ){
    if( box instanceof FunctionBox ){
        ((FunctionBox) box).setX0(A.x);
        ((FunctionBox) box).setY0(A.y);
        ((FunctionBox) box).setX1(B.x);
        ((FunctionBox) box).setY1(B.y);
    }
}
```

en JParametrosCaja.java:

```
public JParametrosCaja( Point puntoA, Point puntoB, Program programa, Box box )...
public void actionBotonCerrarAceptar(){
    this.puntoARet.x = (int)normalizarPunto(punto2).x;
    this.puntoARet.y = (int)normalizarPunto(punto2).y;
    this.puntoBRet.x = (int)normalizarPunto(punto3).x;
    this.puntoBRet.y = (int)normalizarPunto(punto3).y;
    this.programa.setBox( this.box, puntoARet, puntoBRet );
    frameVentana.dispose();
}
```

11.2.4 GUI bolas de poder (función constante)

Si estábamos usando la GUI con Yaks-RAL y poníamos al menos 3 "bolas de energia" a una rueda, ya no se mueve!!

El problema es que el Yaks (simulador) no acepta más de `valor 10` ó `valor -9` para el motor (adelante ó atrás)... Es decir, si le mando `10` o `-9` anda, si le mando `11` o `-10` ya no... En valores normalizados (desde la GUI), para adelante anda hasta valor 54 (54%), desde 55 en adelante ya deja de andar... Para atrás normalizadado anda hasta valor -49 (-49%), desde -50 en adelante ya deja de andar...

Para arreglar esto, modificamos la función `RAL::desNormalizarMotores()` del Yaks para que sature en los valores antes mencionados.

En el RAL-Khepera no tiene este problema, entonces los valores desnormalizados van entre `-20:20`, o normalizados entre `-100%:100%`

11.2.5 Core Finalización

Había problemas con el manejo de señales de finalización (`SIGINT` y `SIGTERM`) del Core junto con la RAL. Un ejemplo de este problema es los que pasaba en 11.2.6.

Al prinicipio sólo atendíamos la señal `SIGINT`. Luego incorporámos la atención de `SIGTERM`. La señal `SIGINT` (*interrupt key signal*), es una señal de atención interactiva, generalmente generada por la teclas `Ctrl+C` en la consola de ejecución, pero que también puede ser enviada por otro programa. La señal `SIGTERM` (*termination signal*), es una señal de terminación enviada por el comando `kill`, pero que también puede ser enviada por otro programa.

También había problemas de entrelazamiento de atención de señales, en particular con RALs que manejan varios threads como RAL-ExaBot, que hacía que los procesos no terminaran en el orden correcto ni de forma correcta. Para solucionar esto hubo que arreglar varias cosas:

- **Core:**
  - La llamada a la función `inicializarRAL()` la hacemos lo más arriba posible (o lo más antes posible) para que, en casos como RAL-ExaBot que tiene varios threads, comparta lo menos posible de memoria con el proceso *padre* Core, como ser *fileDescriptors*, etc. De lo contrario aumenta la probabilidad de que sucedan cosas indeseadas.

  - También por lo anterior, pusimos la llamada a la atención de señales inmediatamente después de `inicializarRAL()`, para que la atención de la señal del Core esté separada del haber levantado los procesos necesarios para la RAL. De lo contrario, sucedía que inmediatamente la señal era atendida también por los procesos de la RAL y en algunos casos, como RAL-ExaBot, esto resultaba en un comportamiento de finalización no deseado.
  - También agregamos para completitud de casos además de `SIGINT`, la atención de la señal `SIGTERM` para que también atienda a la señal del comando de sistema `kill`.

- **RAL-ExaBot:** ver bugs arreglados en 11.2.6.

11.2.6 RAL ExaBot Threads-Signals

Al finalizar el RAL, cuando deteníamos la ejecución del comportamiento desde la GUI, los motores seguían andando, es decir, no les asignaba valor 0 (cero) para detenerlos.

El problema era con el manejo de los procesos `core` (*padre*) y `udp_receive` - `udp_send` (*hijos*). Al enviar las señales de finalización para que el Core las atendiera como `Ctrl+C`, `kill -int Core` o `kill Core` le mandaba la señal a los 3 procesos (padre y 2 hijos), entonces `udp_send` nunca terminaba y nunca llegaba a enviar 0 (cero) a los motores.

Para solucionar esto hubo que arreglar varias cosas:

- **Core:** ver bugs arreglados en 11.2.5.
- **RAL-ExaBot:**
  - Hicimos que los procesos *hijos* (`udp_receive` y `udp_send`) del *padre* RAL-ExaBot (`core`) no dependan de la misma consola, los desatacheamos. Para esto utilizamos el comando de linux *setsid* mediante la función de C `setsid()`, con lo que desatacheamos el hijo del padre. Es para que el hijo no dependa de la misma consola. Es decir, no muera con el `Ctrl+C`, `kill -int Core` o `kill Core` del padre (Core).
    El comando *setsid* crea una nueva sesión para el proceso llamador, quedando como único proceso en este nuevo grupo de procesos.
  - En `udp_send()` agregamos:

```
* signal(SIGINT,terminar_udp_send);  // configurar la rutina de atención de SIGINT
* signal(SIGTERM,terminar_udp_send); // configurar la rutina de atención de SIGTERM
                                     // que no hace nada, pero atiende:
                                     //     void terminar_udp_send(int sig){exit(sig);}
```

  - En `udp_receive()` agregamos:

```
* signal(SIGINT,terminar_udp_receive);  // configurar la rutina de atención de SIGINT
* signal(SIGTERM,terminar_udp_receive); // configurar la rutina de atención de SIGTERM
                                        // que no hace nada, pero atiende:
                                        //     void terminar_udp_receive(int sig){exit(sig);}
```

Chapter 12

Referencias

Bibliography

[1] *XML: Extensible Markup Language*. ????????????? http://?????????????????. (Cited on pages 32 and 33.)

[2] *DOM: Document Object Model*. http://xerces.apache.org/xerces-c/api-3.html, http://www.w3.org/DOM. (Cited on page 40.)

[3] *Khepera Documentation*. http://ftp.k-team.com/khepera/documentation. (Cited on page 18.)

[4] *Khepera User Manual*. http://ftp.k-team.com/khepera/documentation/KheperaUserManual.pdf. (Cited on pages 18 and 19.)

[5] *Khepera Radio Base User Manual*. http://ftp.k-team.com/khepera/documentation/RadioBaseManual.pdf. (Cited on page 18.)

[6] *Khepera Radio Turret User Manual*. http://ftp.k-team.com/khepera/documentation/RadioTurretManual.pdf. (Cited on page 18.)

[7] *The C Book: Signal handling*. http://publications.gbdirect.co.uk/c_book/chapter9/signal_handling.html. (Cited on page 12.)

[8] *Pyro: A python-based versatile programming environment for teaching robotics*. D.S. Blank, D. Kumar, L. Meeden, and H. Yanco. Journal on Educational Resources in Computing (JERIC), Special issue on robotics in undergraduate education. Part 2, 4(3):115, 2004. (Not cited.)

[9] *NQC: Not Quite C*. Baum. http://bricxcc.sourceforge.net/nqc. (Not cited.)

[10] *brickOS*. Markus. http://brickos.sourceforge.net. (Not cited.)

[11] *leJOS: Java for LEGO Mindstorms*. J. Solorzano. http://lejos.sourceforge.net. (Not cited.)

[12] *Microsoft Robotics Developer Studio*. http://www.microsoft.com/robotics. (Not cited.)

[13] *StarLogo TNG: The Next Generation*. MIT's Scheller Teacher Education Program (STEP). http://education.mit.edu/drupal/starlogo-tng. (Not cited.)

[14] *Squeak EToys*. http://www.squeakland.org. (Not cited.)

[15] *Scratch*. http://scratch.mit.edu. (Not cited.)

[16] *RoboLab*. Tufts University. http://www.ceeo.tufts.edu/robolabatceeo. (Not cited.)

[17] *Vehicles: Experiments in Synthetic Psychology*. V. Braitenberg, MIT Press, Cambridge, 1986. (Not cited.)

[18] *Behavior-Based Robotics*. R. C. Arkin, MIT Press, Cambridge, 1998. (Not cited.)