

PROGRAMACIÓN DE ROBOTS

Robots móviles:

Programación de alto nivel para robots móviles

Programación de alto nivel para robots móviles

- 1.- Software robótico: ¿por qué es necesario?
- 2.- Ejemplo de librería: Mobile Robot Programming Toolkit
- 3.- Ejemplo de framework: ROS
- 4.- Otros

Tema 2.- Programación de alto nivel para robots móviles

- 1.- Software robótico: ¿por qué es necesario?
- 2.- Ejemplo de librería: Mobile Robot Programming Toolkit
- 3.- Ejemplo de framework: ROS
- 4.- Otros

1.- Software Robótico: ¿por qué es necesario?

Queremos construir nuestro robot, orientado a la investigación de robótica de rescate en interiores ¿Cómo lo haríamos?

- Posiblemente, compraríamos una base móvil...



Pioneer P3-DX
<http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx>

- Añadiríamos más sensores...



Kinect
<http://www.xbox.com/es-ES/Xbox360/Accessories/Kinect/>



Láser Hokuyo UTM-30LX
http://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html

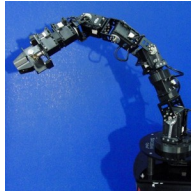


Cámara termal
http://www.thermoteknix.com/content/english/thermal_imaging_ir_products/miricle_thermal_imaging_ir_cameras/index.html

1.- Software Robótico: ¿por qué es necesario?

Queremos construir nuestra robot, orientado a la investigación de robótica de rescate en interiores ¿Cómo lo haríamos?

- Puede que añadiéramos actuadores...



Cyton Gamma 300
http://www.robai.com/products.php?prdt_id=1

- Y necesitaríamos un PC y otros dispositivos de entrada...



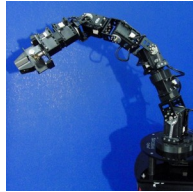
Sony Vaio-S
http://www.sony.es/product/portatil-vaio-s?cid=14001411&s_kwcid=vaio%20pc|18328440180



Joystick Logitech Extreme 3D Pro
<http://www.logitech.com/en-us/gaming/joysticks/extreme-3d-pro>

1.- Software Robótico: ¿por qué es necesario?

¿Y ahora qué hacemos con todo esto :S?



1.- Software Robótico: ¿por qué es necesario?

Tendríamos que resolver los siguientes problemas:

- Conectar todos los dispositivos al PC y/o a la plataforma móvil. No es trivial: por ejemplo, no todos los drivers están disponibles para todos los sistemas operativos.
 - Diseñar nuestra aplicación, de manera que permita recoger los datos de todos los sensores, procesarlos según nuestros algoritmos, y enviarlos a los actuadores para realizar la tarea deseada.
 - Diseñar por tanto las comunicaciones (puede que en tiempo real) entre los distintos elementos de la aplicación, que es una tarea compleja.
 - Programar las herramientas gráficas y de simulación, elementos muy importantes a la hora de la investigación.
-

1.- Software Robótico: ¿por qué es necesario?

Tendríamos que resolver los siguientes problemas:

- Conectar todos los dispositivos al PC y/o a la plataforma móvil. No es trivial: por ejemplo, no todos los drivers están disponibles en todos los sistemas operativos.
- Diseñar nuestra aplicación, de manera que permita recoger los datos de todos los sensores, **procesarlos según nuestros algoritmos**, y enviarlos a los actuadores para realizar la tarea deseada.
- Diseñar por tanto las comunicaciones (puede que en tiempo real) entre los distintos elementos de la aplicación, que es una tarea compleja.
- Programar las herramientas gráficas y de simulación, elementos muy importantes a la hora de la investigación.

¡Sólo esto es investigación!

1.- Software Robótico: ¿por qué es necesario?

¿Cómo se solía hacer tradicionalmente?

- Se creaba una arquitectura específica para el robot, posiblemente monolítica, que permitiera al robot realizar su tarea.
 - Inicialmente parece la solución más rápida y sencilla, pero a la larga es *inmantenible, inescalable e irreutilizable*:
 - El personal va y viene, no hay un control en el mantenimiento de la arquitectura y la gente acaba programando a su manera y/o de cualquier forma.
 - Añadir o sustituir un dispositivo nuevo (el hardware evoluciona continuamente) es complicado, porque hay que tocar directamente muchos elementos de la arquitectura.
 - Reutilizar código (por ejemplo, de otros miembros del equipo) no es sencillo, y perdemos tiempo repitiendo lo mismo.
-

1.- Software Robótico: ¿por qué es necesario?

¿Cómo se puede solucionar el problema?

- Sistemas operativos robóticos (solución inicial)
 - Librerías robóticas: conjunto de librerías que se utilizan para desarrollar programas.
 - Frameworks: conjuntos de librerías y paradigmas que organizaban el desarrollo de la arquitectura.
 - Establecen la organización de la aplicación robótica, estructurándola en elementos software con un esquema definido a seguir. Esto facilita la mantenibilidad, el escalado y la reusabilidad del código.
 - Aportan el sistema de comunicaciones, que es transparente para los programadores.
 - Abstraen la capa hardware.
 - Suelen añadir utilidades gráficas para simulación y recogida de datos.
 - Si se reúne masa crítica suficiente de programadores, la comunidad impulsa nuevos desarrollos.
 - Finalmente, herramientas “CASE” como BABEL, con las que se realiza un diseño de alto nivel que genera automáticamente código.
-

Programación de alto nivel para robots móviles

- 1.- Software robótico: ¿por qué es necesario?
- 2.- Ejemplo de librería: [Mobile Robot Programming Toolkit](#)
- 3.- Ejemplo de framework: ROS
- 4.- Otros

2.- Ejemplo de librería: MRPT

MRPT (*Mobile Robot Programming Toolkit*)

- La MRPT es:
 - Un conjunto de librerías en C++
 - Un conjunto de aplicaciones y herramientas
 - Documentación y soporte
 - Software libre, www.mrpt.org
 - ¿Para qué puede usarse? Para tareas relativas a la robótica móvil
 - Localización
 - SLAM
 - Visión por computador
 - Planificación de movimientos
 - ...
 - Ventajas: eficiencia, reusabilidad, documentación, software libre
-

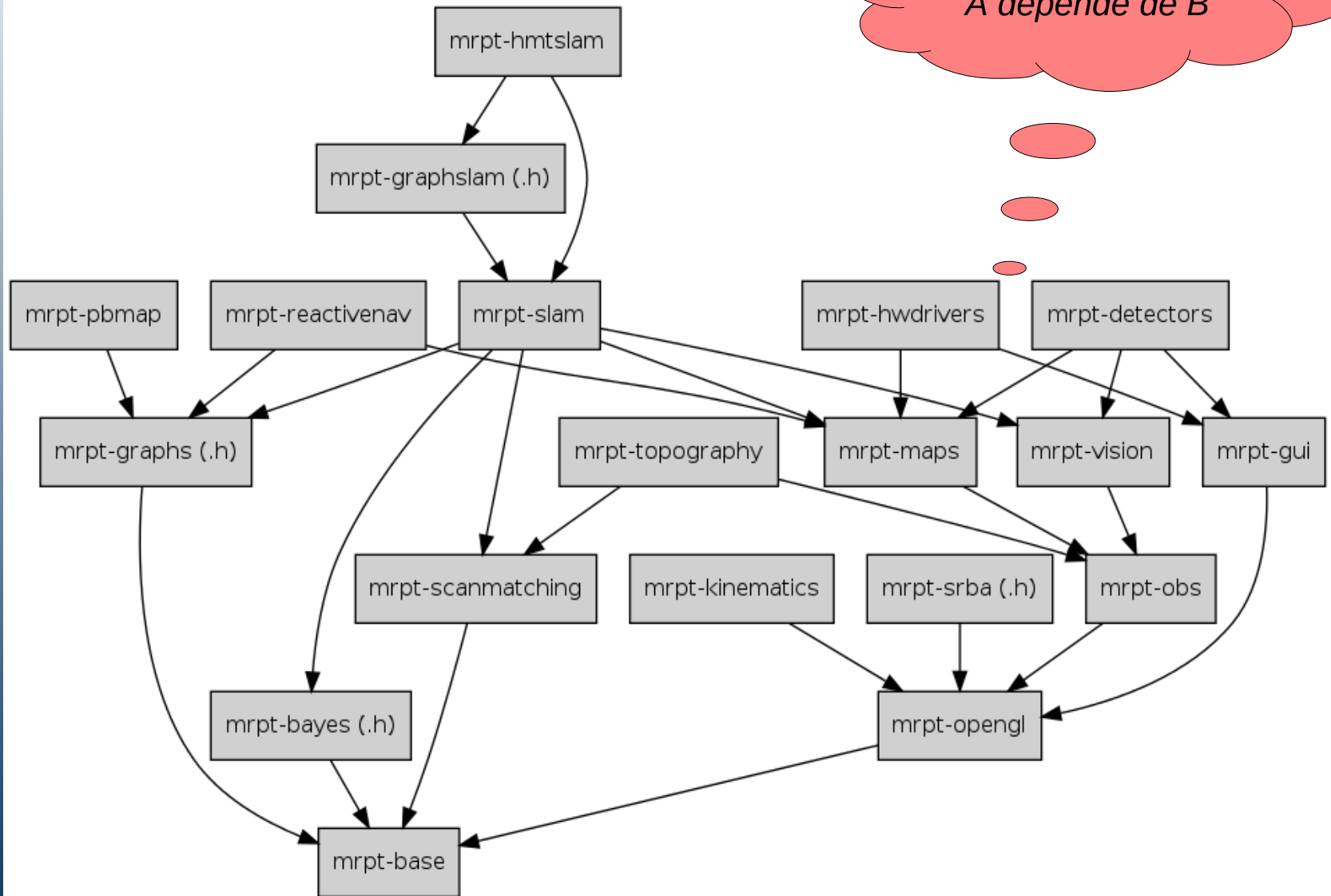
2.- Ejemplo de librería: MRPT

Librerías C++

- Son el elemento más importante de la MRPT
 - Las librerías o módulos agrupan clases y templates C++. Así no es necesario usar toda la MRPT, sino únicamente aquello que nos hace falta.
 - Hay librerías para visión, SLAM, inferencia bayesiana, planificación de caminos, evitación de obstáculos...
-

2.- Ejemplo de librería: MRPT

$A \rightarrow B$ significa que
A depende de B



2.- Ejemplo de librería: MRPT

Aplicaciones

- Son utilidades con las que se pueden realizar operaciones interesantes: simulación de ciertos aspectos, lectura de datos, aplicaciones directas de algoritmos...
 - GUI for Denavit-Hatenberg parameters robot arm design
 - Scene Viewer RD
 - Gridmap navigation dataset generator
 - Rawlog Data Set Viewer
 - ...

2.- Ejemplo de librería: MRPT

Documentación

- Tutoriales: básico, programación, algoritmos...
 - Ejemplos
 - Libro
 - FAQ
 - Datasets
 - Actualizaciones y cambios más o menos constantes
-

2.2.- Ejemplo de librería: MRPT

Instalación

- Última versión estable: 1.3.2 (Noviembre 2015)
 - Plataformas:
 - Windows (binarios precompilados/fuentes)
 - Visual Studio (2008/2010/2012)
 - Linux (fuentes)
 - Recomendado: compilar desde fuentes
 - Comprobar dependencias (wxWidgets, OpenGL...)
 - Cmake también necesario
-

2.- Ejemplo de librería: MRPT

Ejercicio

- Troubleshooting para la versión de Windows :)
 - Comprobar que el path de Windows contiene los directorios bin, lib, libs que cuelgan del directorio raíz de la MRPT
 - Comprobar que en Visual Studio, al pinchar sobre el fichero .cpp y obtener las propiedades, el linkador (“Vinculador”) incluye el path del directorio de las librerías de la MRPT.
 - Puede instalarse en una máquina virtual (Virtual Box); con 2MB de memoria funciona correctamente.

2.- Ejemplo de librería: MRPT

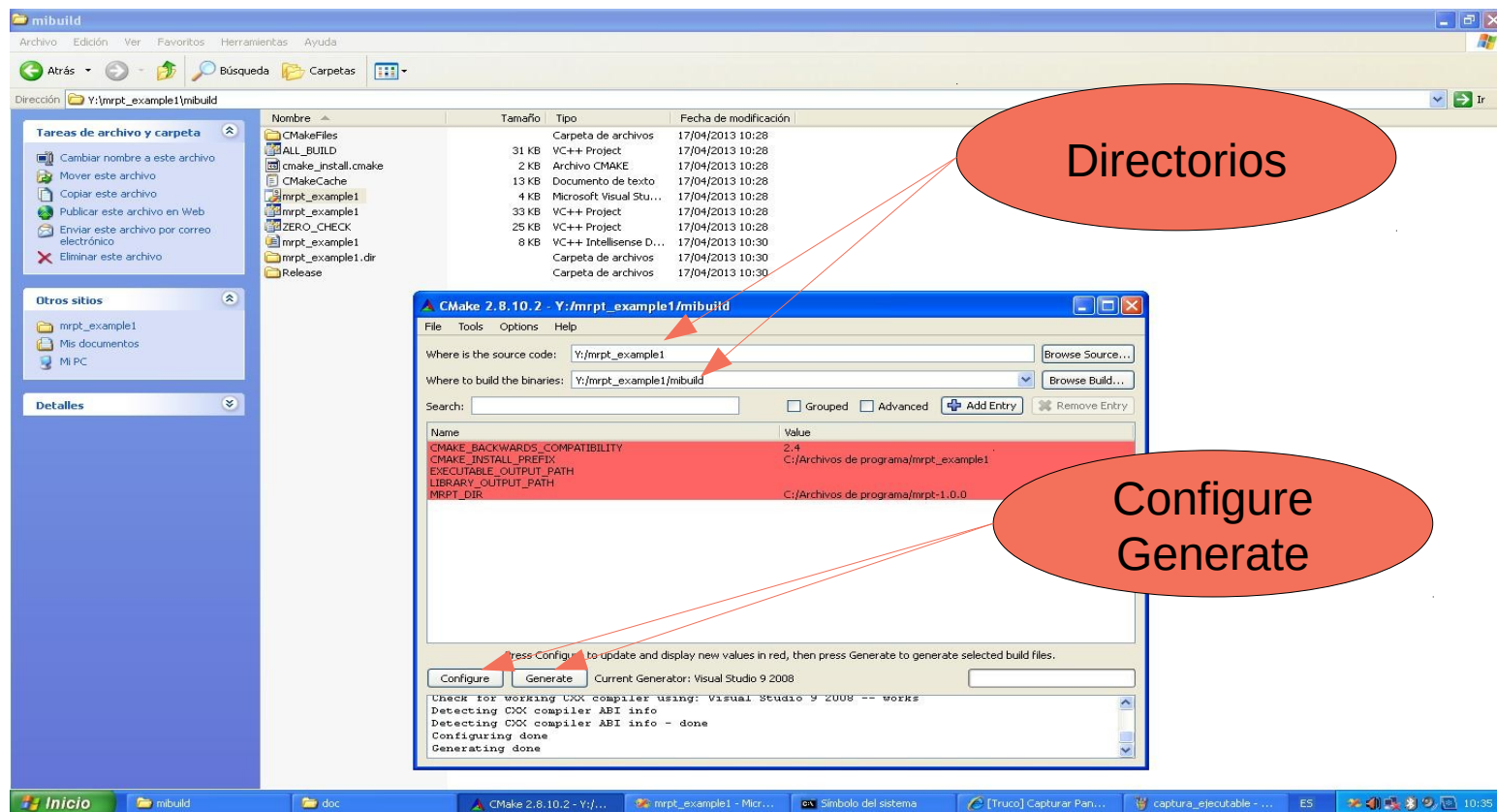
Ejercicio

- Compilar el ejemplo `mrpt_example_1`
 - Copiarlo a vuestro directorio desde `mrpt 1.0.2.` → Documentation directory
 - Primer paso: Cmake
 - Directorios: fuente (donde está el ejemplo) y build (directorio donde se generan los programas)
 - Configure (Visual Studio 2008)
 - Generate (Configuración de desarrollo de Visual C++)
 - Segundo paso: compilar el proyecto en Visual Studio 2008
 - En el directorio build, abrir el fichero *solution* con Visual Studio 2008
 - Compilar: Release, botón derecho sobre el `.cpp`
 - Generar
- Ejecutarlo
 - Llamar desde ventana de comandos

2.- Ejemplo de librería: MRPT

Ejercicio

- Cmake: genera los makefiles para diferentes compiladores y plataformas a partir de ficheros de configuración (CMakeLists.txt) independientes de ambos
- Si CmakeLists.txt no está creado, hay que crearlo



2.- Ejemplo de librería: MRPT

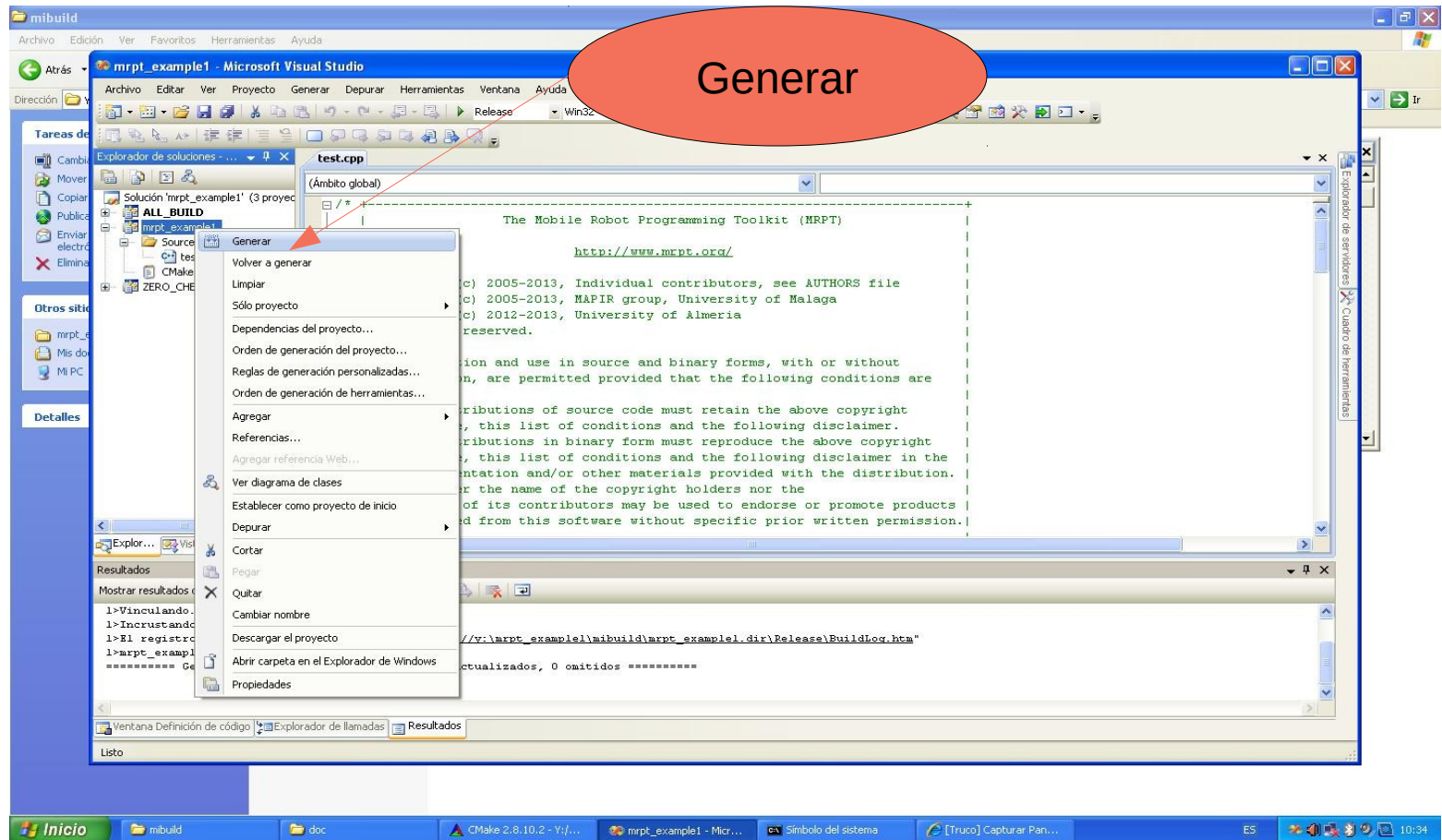
Ejercicio

- Cmake: genera los makefiles para diferentes compiladores y plataformas a partir de ficheros de configuración (CMakeLists.txt) independientes de ambos
- Si CmakeLists.txt no está creado, hay que crearlo

```
PROJECT(mrpt_example1)
CMAKE_MINIMUM_REQUIRED(VERSION 2.4)
if(COMMAND cmake_policy)
    cmake_policy(SET CMP0003 NEW) # Required by CMake 2.7+
endif(COMMAND cmake_policy)
# -----
# The list of "libs" which can be included can be found in:
# http://www.mrpt.org/Libraries
#
# The dependencies of a library are automatically added, so you only
# need to specify the top-most libraries your code depends on.
# -----
FIND_PACKAGE( MRPT REQUIRED base) # WARNING: Add all the MRPT libs used by your program: "gui",
"obs", "slam", etc.
# Declare the target (an executable)
ADD_EXECUTABLE(mrpt_example1
    test.cpp
)
TARGET_LINK_LIBRARIES(mrpt_example1 ${MRPT_LIBS})
# Set optimized building:
IF(CMAKE_COMPILER_IS_GNUCXX AND NOT CMAKE_BUILD_TYPE MATCHES "Debug")
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O3 -mtune=native")
ENDIF(CMAKE_COMPILER_IS_GNUCXX AND NOT CMAKE_BUILD_TYPE MATCHES "Debug")
```

2.- Ejemplo de librería: MRPT

Ejercicio: test.cpp



2.- Ejemplo de librería: MRPT

Ejercicio: test.cpp

```
#include <mrpt/base.h>
using namespace mrpt::utils;
using namespace mrpt::poses;
using namespace std;
int main()
{ [...]

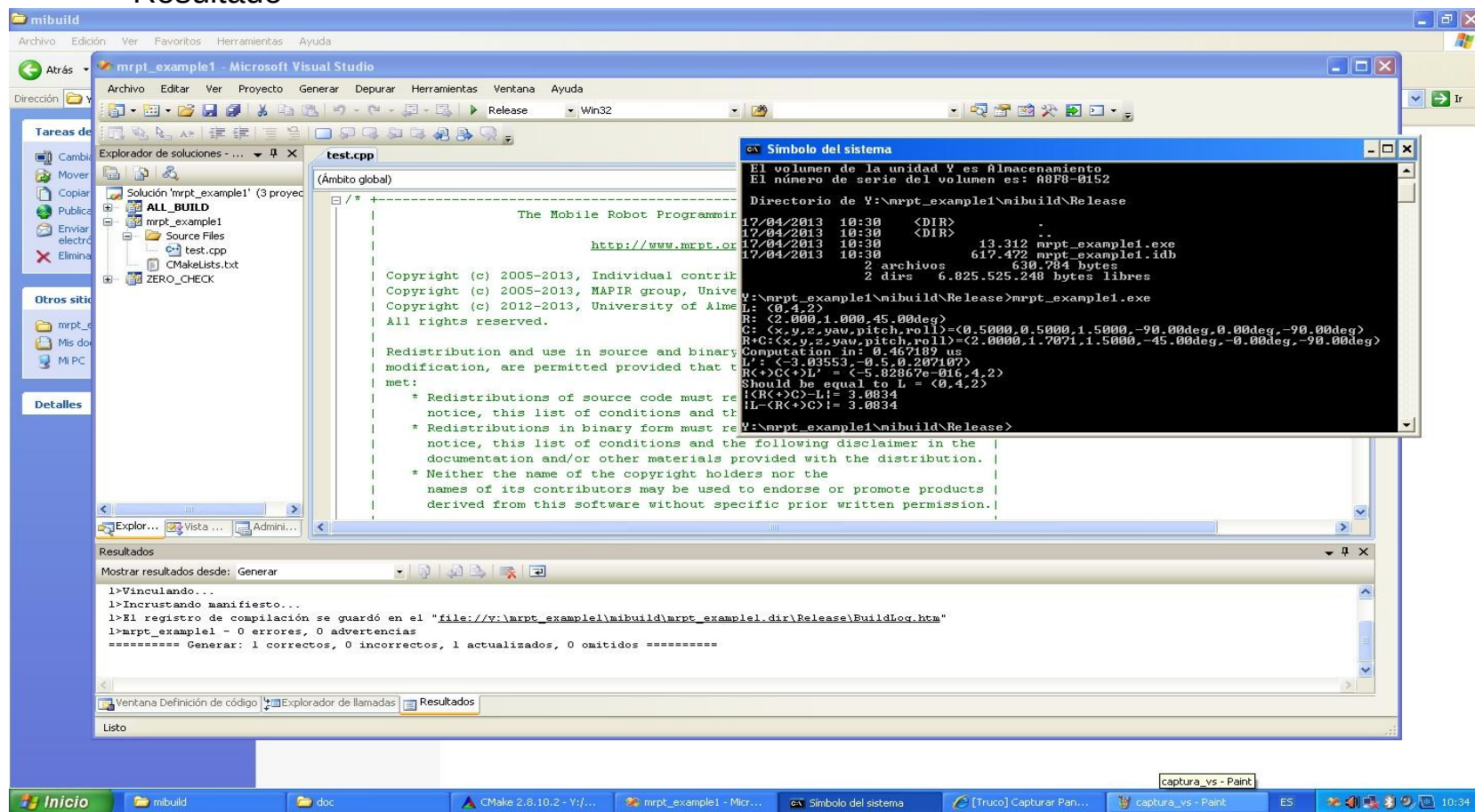
    // The landmark (global) position: 3D (x,y,z)
    CPoint3D L(0,4,2);
    // Robot pose: 2D (x,y,phi)
    CPose2D R(2,1, DEG2RAD(45.0f) );
    // Camera pose relative to the robot: 6D (x,y,z,yaw,pitch,roll).
    CPose3D C(0.5f,0.5f,1.5f ,DEG2RAD(-90.0f),DEG2RAD(0),DEG2RAD(-90.0f) );
    // TEST 1. Relative position L' of the landmark wrt the camera
    // -----
    cout << "L: " << L << endl;
    cout << "R: " << R << endl;
    cout << "C: " << C << endl;
    cout << "R+C:" << (R+C) << endl;
    //cout << (R+C).getHomogeneousMatrix();
    CPoint3D L2;
    CTicTac tictac;
    tictac.Tic();
    size_t i,N = 10000;
    for (i=0;i<N;i++)
        L2 = L - (R+C);
    cout << "Computation in: " << 1e6 * tictac.Tac()/((double)N) << " us" << endl;
    cout << "L': " << L2 << endl;
    // TEST 2. Reconstruct the landmark position:
    // -----
    CPoint3D L3 = R + C + L2;
    cout << "R(+)C(+)L' = " << L3 << endl;
    cout << "Should be equal to L = " << L << endl;
    // TEST 3. Distance from the camera to the landmark
    // -----
    cout << "|R(+)C)-L|= " << (R+C).distanceTo(L) << endl;
    cout << "|L-(R(+)C)|= " << L-(R+C).distanceTo(L) << endl;
    return 0;
}
```

[...] }

2.- Ejemplo de librería: MRPT

Ejercicio

– Resultado



programación de alto nivel para robots móviles

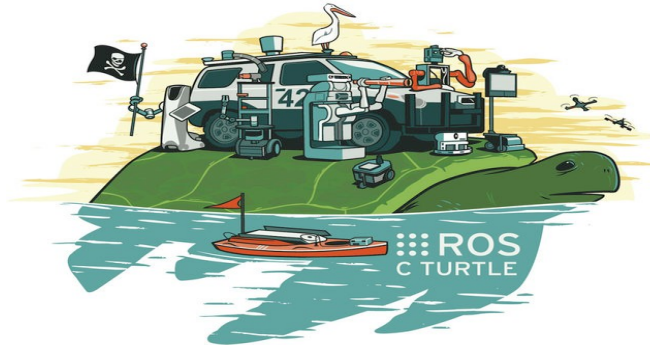
- 1.- Software robótico: ¿por qué es necesario?
- 2.- Ejemplo de librería: Mobile Robot Programming Toolkit
- 3.- Ejemplo de framework: ROS
- 4.- Otros

3.- Ejemplo de framework: ROS

- Nosotros vamos a usar un framework concreto muy utilizado en la actualidad: ROS
 - Existen otros frameworks bastante conocidos, que posteriormente comentaremos
-

3.- Ejemplo de framework: ROS

*ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides **hardware abstraction**, **device drivers**, **libraries**, **visualizers**, **message-passing**, package management, and more. ROS is licensed under an open source, BSD license.*



ROS.org

ROS
<http://www.ros.org>



Willow Garage
<http://www.willowgarage.com/>

3.- Ejemplo de framework: ROS

Quigley M., Gerkey B., Conley K., Faust J., Foote T., Leibs J., Berger E., Wheeler R., Ng A. (2009) *ROS: an open-source Robot Operating System*, International Conference on Robotics and Automation.

- Peer-to-peer
 - En tiempo de ejecución hay un cjto. de procesos conectados, puede que distribuidos.
 - No hay un servidor central de datos.
 - Hay un servidor de nombres para que los procesos se “encuentren”.
 - Multilenguaje
 - C++, python, Octave, Lisp
 - Personas distintas prefieren lenguajes distintos, permite optimizar habilidades.
 - Thin
 - La funcionalidad (drivers, algoritmos) se incluye en librerías sin dependencias de ROS.
 - Pequeños ejecutables que muestran las funcionalidades de la librería a ROS
 - Facilita la reutilización
 - Gratuita y de código abierto
 - Basada en herramientas
 - Pequeñas herramientas que construyen los componentes de ROS
 - Usas lo que necesitas
-

3.- Ejemplo de framework: ROS

- Versión actual: Melodic (vamos a usar una anterior: Indigo)
 - Sistemas Operativos
 - Ubuntu
 - Experimentales
 - OS X, Fedora, Gentoo, OpenSuse, Debian, Arch Linux, Windows...
 - Lenguajes
 - C++, Python, Lisp, Octave
 - Experimentales: JAVA, Lua
-

3.- Ejemplo de framework: ROS

Instalación de ROS según su wiki:

- <http://wiki.ros.org/ROS/Installation>
- Una vez instalado, hay que seguir los tutoriales: www.ros.org/wiki

3.- Ejemplo de framework: ROS

Instalación de ROS según su wiki

- <http://wiki.ros.org/ROS>

**¡¡Instalación de ROS en VirtualBox con Ubuntu
14.04 :)!!**

<http://nootrix.com/downloads/>

Sobre ella he creado una nueva para usar Arduino

3.- Ejemplo de framework: ROS

Ejercicio

- Instalación de la máquina virtual en la máquina del laboratorio
 - Copiar la máquina virtual al escritorio
 - Hacer doble click sobre el fichero de la máquina virtual. Importar sin tocar ninguna opción.
 - En las máquinas nuevas del lab, puede ser necesario desactivar USB y CD (posiblemente por el filtro de Arduino que ya trae la máquina)
 - Al principio parece que tarda mucho, pero se instala en varios minutos.

3.- Ejemplo de framework: ROS

Ejercicio

- Configuración del entorno de trabajo de ROS en máquina virtual de Nootrix con ROS Indigo
 - Hay que reconfigurar el teclado en castellano:
 - Settings → Keyboard → Text Entry → Input Sources to Use → + → Spanish
 - Si no se activa, ir a la izquierda de la barra superior y escoger idioma
 - La clave de sudo es *viki*
 - En el menú de la máquina virtual, activar el portapapeles como bidireccional, para poder pasar texto entre la máquina virtual y el host (otras formas de comunicación entre ambos son red, usb, carpeta compartida)
 - Carpetas compartidas:
 - Configuración VirtualBox: añadir Y:\ (automontar, permanente)
 - ROS (es posible que haya que montar la carpeta en cada sesión):
 - » Crear carpeta Y_DRIVE en /home/viki
 - » `sudo mount -t vboxsf Y_DRIVE ~/Y_DRIVE`
 - » Para copiar/pegar en la carpeta, lanzar *nautilus* desde terminal

3.- Ejemplo de framework: ROS

Ejercicio

- Configuración del entorno de trabajo de ROS en máquina virtual de Nootrix con ROS Indigo
 - Comprobar que las variables de entorno existen: `export | grep ROS`

```
viki@c3po: ~  
viki@c3po:~$ export | grep ROS  
declare -x ROSLISP_PACKAGE_DIRECTORIES=""  
declare -x ROS_DISTRO="indigo"  
declare -x ROS_ETC_DIR="/opt/ros/indigo/etc/ros"  
declare -x ROS_MASTER_URI="http://localhost:11311"  
declare -x ROS_PACKAGE_PATH="/opt/ros/indigo/share:/opt/ros/indigo/stacks"  
declare -x ROS_ROOT="/opt/ros/indigo/share/ros"  
viki@c3po:~$
```

3.- Ejemplo de framework: ROS

Ejercicio

- Configuración del entorno de trabajo de ROS en máquina virtual de Nootrix con ROS Indigo
 - En los tutoriales, veréis que algunas tareas se pueden hacer usando *roscpp* y *catkin*. Estos son los dos métodos disponibles para organizar y construir código ROS; las diferencias radican en:
 - *roscpp* es más sencillo de usar y simple. Sólo puede usarse hasta Fuerte.
 - *catkin* es más sofisticado pero más flexible, sobre todo para integrar código externo. Está disponible a partir de Fuerte.
 - Más información en http://wiki.ros.org/catkin_or_roscpp.
 - Como nuestra versión es Indigo, vamos a usar *catkin*.

3.- Ejemplo de framework: ROS

Los puntos que están en verde
ya están ejecutados en la máquina
virtual de la asignatura

Ejercicio

- Configuración del entorno de trabajo de ROS en máquina virtual de Nootrix con ROS Indigo
- Preparación del workspace catkin
 - La instalación de Nootrix ya nos trae un directorio `~/catkin_ws/`
 - Comprobar qué subdirectorios trae por defecto
 - Inicializar el espacio de trabajo, ejecutando en el directorio `src:` `catkin_init_workspace`
 - » Si no se hace así, al hacer `catkin_make` (siguiente paso) dará error, y no será posible hacer build de ningún paquete.
 - » Soluciones: reinstalar la máquina virtual, o borrar el directorio `catkin_ws` entero y crear el espacio de trabajo siguiendo los tutoriales de ROS
- Construir el espacio de trabajo
 - Ejecutar en el directorio raíz del workspace: `catkin_make`
 - Comprobar que se han creado los directorios `devel` y `build`
 - Hacer el siguiente source: `source devel/setup.bash`
 - Comprobar que la variable `ROS_PACKAGE_PATH` incluye nuestro directorio:
`echo $ROS_PACKAGE_PATH`

```
/home/viki/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/stacks
```

3.- Ejemplo de framework: ROS

Hay cuatro conceptos fundamentales a la hora de comprender ROS (Quigley M., Gerkey B., Conley K., Faust J., Foote T., Leibs J., Berger E., Wheeler R., Ng A. (2009) *ROS: an open-source Robot Operating System*, International Conference on Robotics and Automation.)

- **Nodos**
 - Procesos que realizan los cálculos, puede que distribuidos
 - En tiempo de ejecución una aplicación ROS es un conjunto de nodos.
 - **Mensajes**
 - Comunicaciones entre nodos.
 - Son tipos de datos estructurados; pueden anidarse.
 - Se definen en IDL, en un fichero de texto
 - **Topics**
 - Canales de comunicación por los que circulan mensajes entre nodos.
 - Los nodos publican en topics (publishers); se suscriben a los topics que les interesan (subscribers). Publishers y subscribers no conocen la existencia de otros.
 - Habitualmente, se forman grafos complejos.
 - **Servicios**
 - Canales de comunicación para comunicaciones síncronas entre nodos.
 - Formados por un nombre y dos mensajes: petición (request) y respuesta (response).
 - Un nodo hace una petición a otro nodo y recibe una respuesta.
-

3.- Ejemplo de framework: ROS

Hay cuatro conceptos fundamentales a la hora de comprender ROS (Quigley M., Gerkey B., Conley K., Faust J., Foote T., Leibs J., Berger E., Wheeler R., Ng A. (2009) *ROS: an open-source Robot Operating System*, International Conference on Robotics and Automation.)

- **Nodos**
 - *Procesos que realizan los cálculos, puede que distribuidos*
 - *En tiempo de ejecución una aplicación ROS es un conjunto de nodos.*
 - Mensajes
 - Comunicaciones entre nodos.
 - Son tipos de datos estructurados; pueden anidarse.
 - Se definen en IDL, en un fichero de texto
 - **Topics**
 - *Canales de comunicación por los que circulan mensajes entre nodos.*
 - *Los nodos publican en topics (publishers); se suscriben a los topics que les interesan (subscribers). Publishers y subscribers no conocen la existencia de otros.*
 - *Habitualmente, se forman grafos complejos.*
 - Servicios
 - Canales de comunicación para comunicaciones síncronas entre nodos.
 - Formados por un nombre y dos mensajes: petición (request) y respuesta (response).
 - Un nodo hace una petición a otro nodo y recibe una respuesta.
-

3.- Ejemplo de framework: ROS

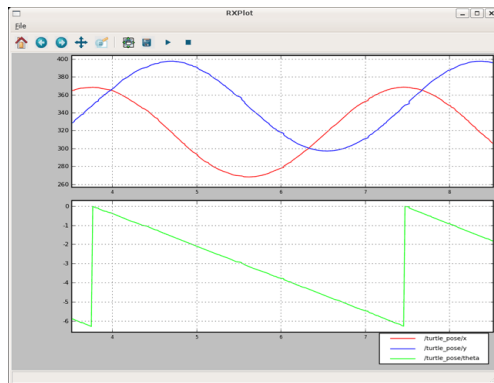
A la hora de trabajar con ROS, puede hacerse a tres niveles

- Nivel gráfico: recogida de datos, simulación, diagnóstico
 - Diagnóstico
 - Recogida de datos
 - Simulación
 - Sistema de ficheros y mensajes
 - Nodos
 - Paquetes
 - Stacks
 - Mensajes
 - Topics
 - Comunidad
 - Repositorio
 - Wiki
 - Distribuciones
-

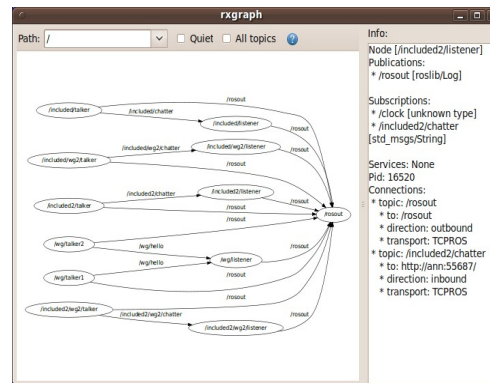
3.- Ejemplo de framework: ROS

A la hora de trabajar con ROS, puede hacerse a tres niveles

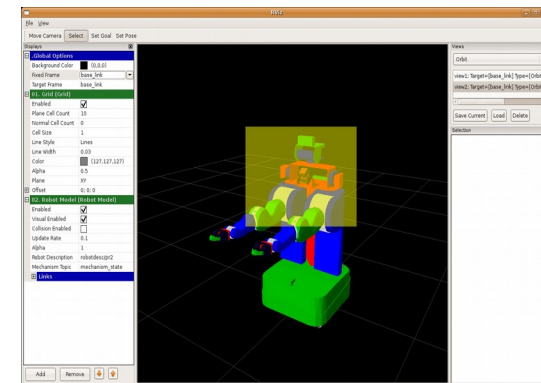
- Nivel gráfico: recogida de datos, simulación, diagnóstico
 - Diagnóstico
 - Recogida de datos
 - Simulación



rxplot



rxgraph



rviz

3.- Ejemplo de framework: ROS

A la hora de trabajar con ROS, puede hacerse a tres niveles

- Nivel gráfico: recogida de datos, simulación, diagnóstico
 - Diagnóstico
 - Recogida de datos
 - Simulación
- Sistema de ficheros y mensajes
 - Nodos
 - Paquetes
 - Stacks
 - Mensajes
 - Topics
- Comunidad
 - Repositorio
 - Wiki
 - Distribuciones

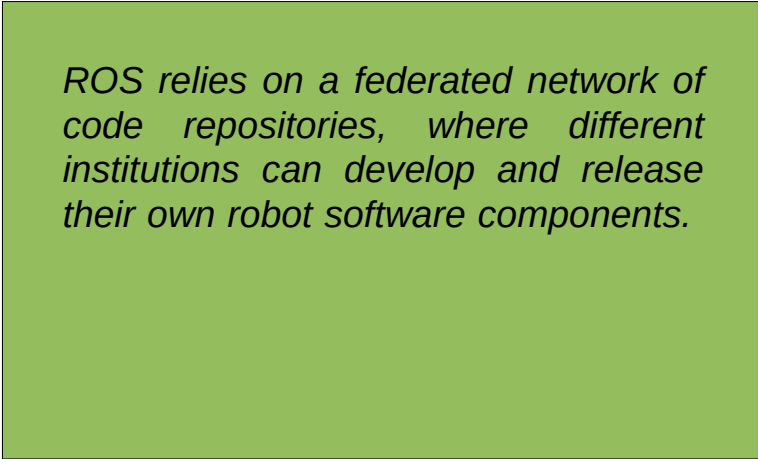


Lo veremos en el siguiente punto

3.- Ejemplo de framework: ROS

A la hora de trabajar con ROS, puede hacerse a tres niveles

- Nivel gráfico: recogida de datos, simulación, diagnóstico
 - Diagnóstico
 - Recogida de datos
 - Simulación
- Sistema de ficheros y mensajes
 - Nodos
 - Paquetes
 - Stacks
 - Mensajes
 - Topics
- Comunidad
 - Repositorio
 - Wiki
 - Distribuciones



ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.

3.- Ejemplo de framework: ROS

A la hora de trabajar con ROS, puede hacerse a tres niveles

- Nivel gráfico: recogida de datos, simulación, diagnóstico
 - Diagnóstico
 - Recogida de datos
 - Simulación
- Sistema de ficheros y mensajes
 - Nodos
 - Paquetes
 - Stacks
 - Mensajes
 - Topics
- Comunidad
 - Repositorio
 - Wiki
 - Distribuciones

The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.

3.- Ejemplo de framework: ROS

A la hora de trabajar con ROS, puede hacerse a tres niveles

- Nivel gráfico: recogida de datos, simulación, diagnóstico
 - Diagnóstico
 - Recogida de datos
 - Simulación
- Sistema de ficheros y mensajes
 - Nodos
 - Paquetes
 - Stacks
 - Mensajes
 - Topics
- Comunidad
 - Repositorio
 - Wiki
 - Distribuciones

ROS Distributions are collections of versioned stacks that you can install. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent versions across a set of software.

3.- Ejemplo de framework: ROS

Paquetes y Stacks

- Paquete (*Package*)
 - Nivel más bajo de la organización software ROS.
 - Directorio con fuentes, librerías, makefiles, builds...
 - Herramientas de paquetes: *rospack*, ...

- Stack
 - Paquetes que funcionan juntos
 - Herramientas de stacks: *rostack*,...

3.- Ejemplo de framework: ROS

Paquetes y Stacks

- ***Paquete (Package)***
 - ***Nivel más bajo de la organización software ROS.***
 - ***Directorio con fuentes, librerías, makefiles, builds...***
 - ***Herramientas de paquetes: rospack, ...***
- Stack
 - Paquetes que funcionan juntos
 - Herramientas de stacks: *rosstack*,...

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 2.- Navegar por el sistema de ficheros (*Navigating the ROS Filesystem*)
 - *rospack* y *rostack* proporcionan información sobre paquetes y stacks
 - *rospack help*
 - *rospack find roscpp*
 - *roscd* sólo busca en los directorios de \$ROS_WORKSPACE
 - *roscd* te lleva a \$ROS_WORKSPACE
 - *roscd* directorio/subdirectorio
 - *roscd* log te lleva al directorio de ficheros de log (es necesario haber ejecutado algún programa de ROS antes al menos una vez)
 - *rosls* hace un ls no sólo por el path, sino por paquete, stack o localización.
-

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 2.- Navegar por el sistema de ficheros (*Navigating the ROS Filesystem*)
 - Para obtener las dependencias de primer orden de un paquete:
rospack depends1 beginner_tutorials
 - Para obtener las dependencias anidadas de un paquete:
rospack depends beginner_tutorials
 - Las dependencias también aparecen en el fichero *package.xml*

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 3.- Creación de un paquete ROS (*Creating a ROS Package*)
 - Los paquetes que vamos a crear son catkin, así que deben cumplir una serie de requisitos:
 - Deben incluir un fichero *package.xml* adaptado a catkin, con meta información sobre el paquete.
 - Deben incluir un fichero *CMakeLists.txt* adaptado a catkin.
 - No puede haber más de un paquete por directorio
 - » Por tanto, no es posible tener paquetes anidados ni múltiples paquetes compartiendo el mismo directorio.

```
my_package/  
  CMakeLists.txt  
  package.xml
```

Este sería el paquete catkin más simple

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 3.- Creación de un paquete ROS (*Creating a ROS Package*)
 - Es conveniente crear un espacio de trabajo catkin
 - Este es el aspecto que debe tener un espacio catkin sencillo
 - Nosotros ya lo hemos hecho (comprobar que la estructura es parecida)

```
workspace_folder/      -- WORKSPACE
src/                   -- SOURCE SPACE
  CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
  package_1/
    CMakeLists.txt     -- CMakeLists.txt file for package_1
    package.xml        -- Package manifest for package_1
  ...
  package_n/
    CMakeLists.txt     -- CMakeLists.txt file for package_n
    package.xml        -- Package manifest for package_n
```

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:

http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 3.- Creación de un paquete ROS (*Creating a ROS Package*)
 - *catkin_create_pkg* crea un paquete nuevo, indicando además las dependencias. Debe llamarse en el directorio *src* de nuestro espacio de trabajo catkin.
 - Se crea automáticamente un directorio que almacena una serie de ficheros (*CmakeLists.txt*, *package.xml*) y puede que algunos directorios, según las dependencias establecidas.
 - Ejercicio: crear el paquete *beginners_tutorial* según se indica en el tutorial correspondiente de ROS. Si ya existe, comprobar qué ocurre, y qué estructura de ficheros se ha creado.
-

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 4.- Compilación de un paquete ROS (*Building a ROS Package*)
 - Hacer el build del paquete, en ese caso en el directorio raíz del espacio de trabajo: `catkin_make`
 - Finalmente, añadir el espacio de trabajo al entorno ROS:
`. ~/catkin_ws/devel/setup.bash`
 - Recordar
 - Creación del paquete: `catkin_create_pkg` en `~/catkin_ws/src`
 - Compilación del paquete: `catkin_make` en `~/catkin_ws/`

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 5.- Nodos ROS (*Understanding ROS Nodes*)
 - Un nodo es un ejecutable que usa ROS para comunicarse con otros nodos.
 - Master: servidor de nombres de ROS, para que los nodos se “encuentren”
 - *rosout*: ROS stdout/stderr.
 - *roscore*: Master+*rosout*+parameter server. Sólo puede haber uno corriendo.
 - *rostopic list*: información sobre los nodos que se están ejecutando en ese momento. Al menos debe aparecer *rosout*.
 - *rostopic pub nombre_paquete nombre_nodo*: lanza el nodo, no es necesario conocer el path del paquete. Ctrl-C para pararlo.
 - Ejercicio: comprobar el funcionamiento de *roscore*, *rostopic* y *rostopic pub* según se indica en el correspondiente tutorial
-

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 6.- Topics ROS (*Understanding ROS Topics*)
 - *rqt_graph* crea un grafo dinámico donde aparecen los nodos, los topics, y las relaciones entre ellos.
 - *rostopic*
 - *rostopic echo*: muestra el contenido de un topic.
 - *rostopic list*: lista los topics activos con subscribers y publishers.
 - *rostopic type*: muestra el tipo de los mensajes circunlando por un topic.
 - *rostopic pub*: publica datos en un topic.
 - *rostopic hz*: muestra la frecuencia de publicación de datos en un topic.
 - *rqt_plot*: muestra un gráfico temporal de los datos publicados en un topic.
 - Ejercicio: comprobar el funcionamiento de *rostopic*, *rqt_graph* (ya está instalado) y *rqt_plot* según se indica en el tutorial correspondiente de ROS
-

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 10.- Creación de un publisher y un suscriber C++
 - Tutoriales:
 - Writing a Simple Publisher and Subscriber (C++)
 - Examining the Simple Publisher and Subscriber
 - Crear los ficheros *.cpp* en el directorio *src* del directorio *beginner_tutorials*
 - Modificar *CmakeLists.txt* añadiendo al final

```
include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker beginner_tutorials_generate_messages_cpp

add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener beginner_tutorials_generate_messages_cpp)
```
 - Hacer *catkin_make* en el directorio raíz del espacio de trabajo
-

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 10.- Creación de un publisher y un susbcriber C++
 - Lanzar *roscore* en un terminal
 - Lanzar los ejecutables, cada uno en un terminal aparte
 - *roslaunch* nombre_paquete nombre_nodo_publisher
 - *roslaunch* nombre_paquete nombre_nodo_subscriber
 - Antes de lanzar cada ejecutable, hacer *source ./devel/setup.bash*

3.- Ejemplo de framework: ROS

Una vez instalado, hay que seguir los tutoriales:
http://wiki.ros.org/ROS/Tutorials#Beginner_Level

- 10.- Creación de un publisher y un susbcriber C++
 - Pasos a seguir en el fichero .cpp
 - Si el nodo es un publisher
 - » *init*: aquí se debe incluir el nombre del nodo
 - » *NodeHandle*
 - » *advertise* en un topic, incluyendo mensaje y nombre del topic
 - » Indicar frecuencia de publicación
 - » Publicar
 - Si el nodo es un subscriber
 - » Definir la *callback*
 - » *init*: aquí se debe incluir el nombre del nodo
 - » *NodeHandle*
 - » Suscribirse
 - » *spin*

Programación de alto nivel para robots móviles

- 1.- Software robótico: ¿por qué es necesario?
- 2.- Ejemplo de librería: Mobile Robot Programming Toolkit
- 3.- Ejemplo de framework: ROS
- 4.- Otros

4.- Otros

Existen otros frameworks disponibles:

- Player/Stage
- Microsoft Robotics Developer Studio
- BABEL
- OpenMora
- Coppelia
- YARP (ver transparencias y vídeo web)

4.- Otros

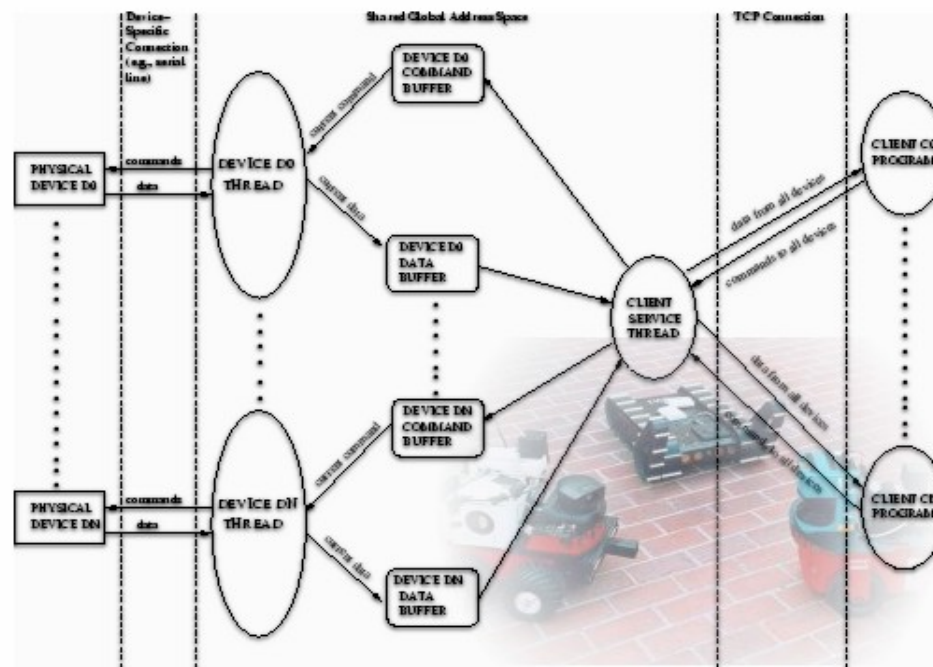
Player/Stage:

- <http://playerstage.sourceforge.net/>
 - Última actualización: 2010
 - Player: interfaz con robots (reales o simulados)
 - Multilenguaje
 - Stage: simulación 2D (muchos) / Gazebo: simulación 3D (pocos)
 - Linux/Mac/Solaris
-

4.- Otros

Player/Stage:

- Player

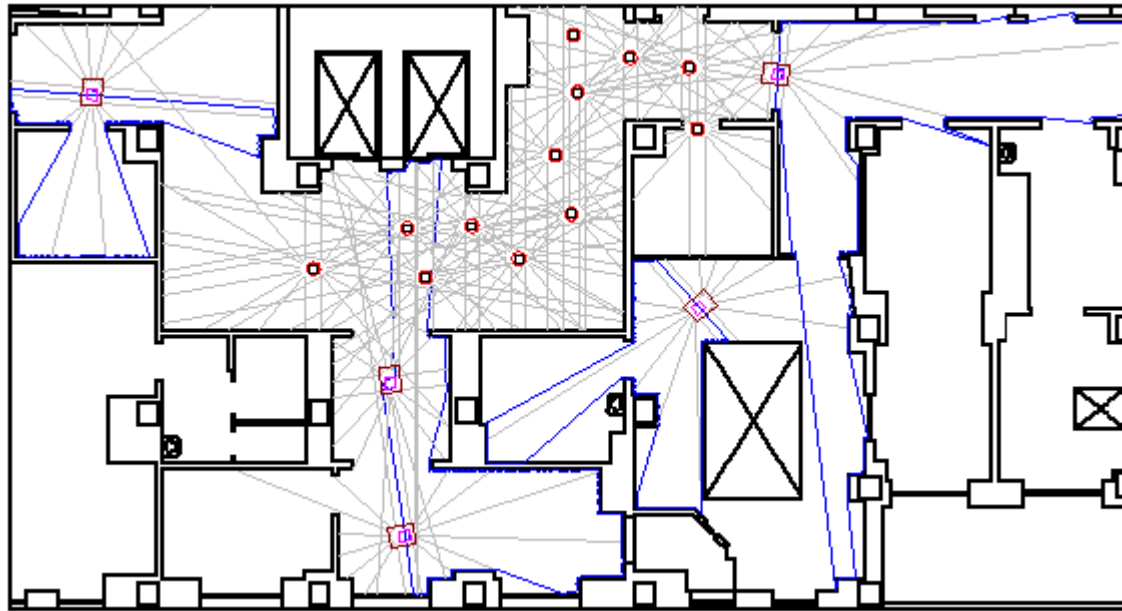


<http://robotics.usc.edu/?l=Projects:playerstagegazebo>

4.- Otros

Player/Stage:

- Stage

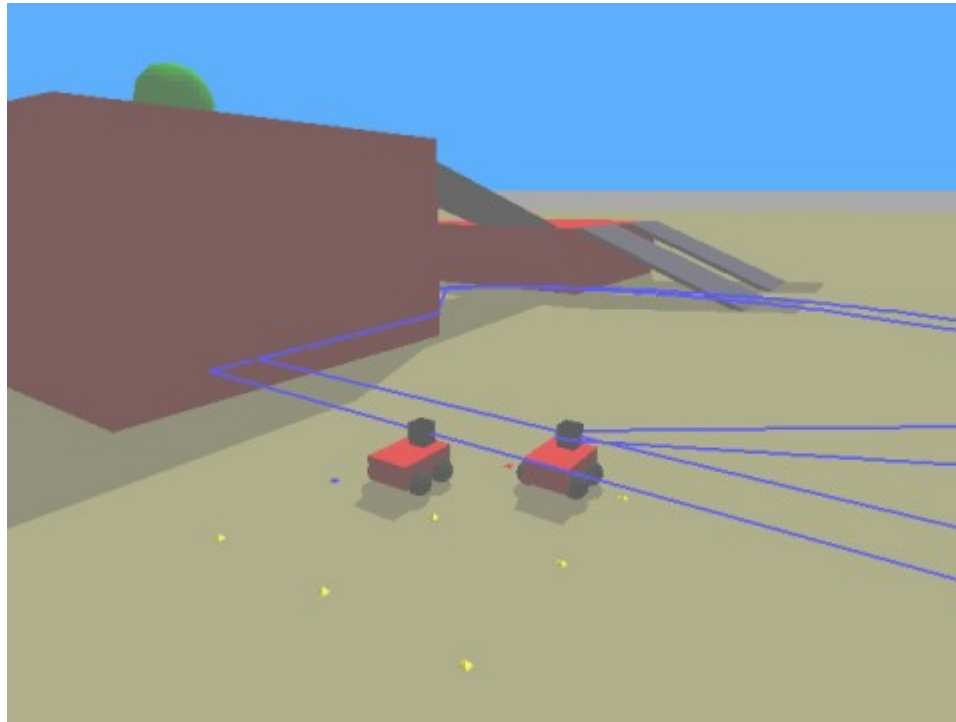


<http://robotics.usc.edu/?l=Projects:playerstagegazebo>

4.- Otros

Player/Stage:

- Gazebo



<http://robotics.usc.edu/?l=Projects:playerstagegazebo>

4.- Otros

Microsoft Robotics Developer Studio:

- <http://www.microsoft.com/robotics/>
 - Windows
 - Lenguaje de programación visual (curva de aprendizaje compleja)
 - C#
-

4.- Otros

BABEL:

- Desarrollado en el Dpto. de Ingeniería de Sistemas y Automática de la UMA (1996-actualidad)
- Software libre
- http://babel.isa.uma.es/babel2/index.php/Main_Page



4.- Otros

BABEL:

- Su objetivo principal es adaptarse a la heterogeneidad propia de las aplicaciones robóticas (código, plataformas hardware, formación y rotación del personal...)
 - Neutral respecto al sistema operativo y al lenguaje de programación (actualmente, implantada para Windows, C/C++ y Java)
 - Permite desarrollar aplicaciones robóticas complejas mediante el desarrollo y la integración de módulos software.
-

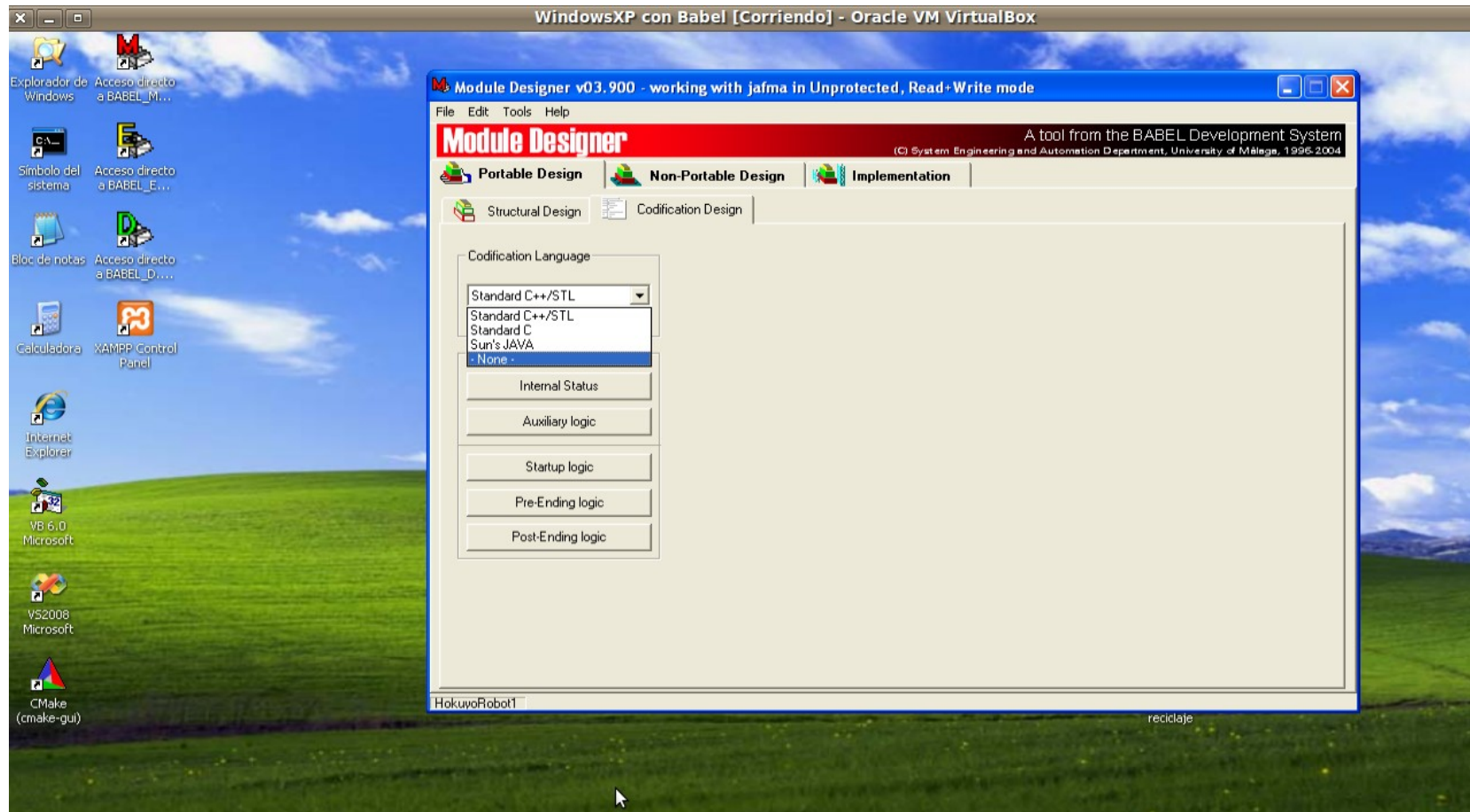
4.- Otros

BABEL:

- Consta de varios elementos
 - Una especificación que indica cómo deben realizarse el diseño (la especificación actual se denomina Aracne, se está evolucionando hacia H)
 - Un diseñador de módulos (BABEL Module Designer), que permite diseñarlos visualmente, y además generar semiautomáticamente parte del código fuente
 - Un generador de módulos (BABEL Generator), que genera el código desde línea de comandos
 - Un gestor de módulos visual (BABEL Execution Manager), que permite organizar y controlar la aplicación robótica completa, puede que distribuida: lanzamiento de módulos, creación de logs...
 - Una herramienta visual (BABEL Debugger) para analizar logs de tiempos: fallos en el cumplimiento de las restricciones...
-

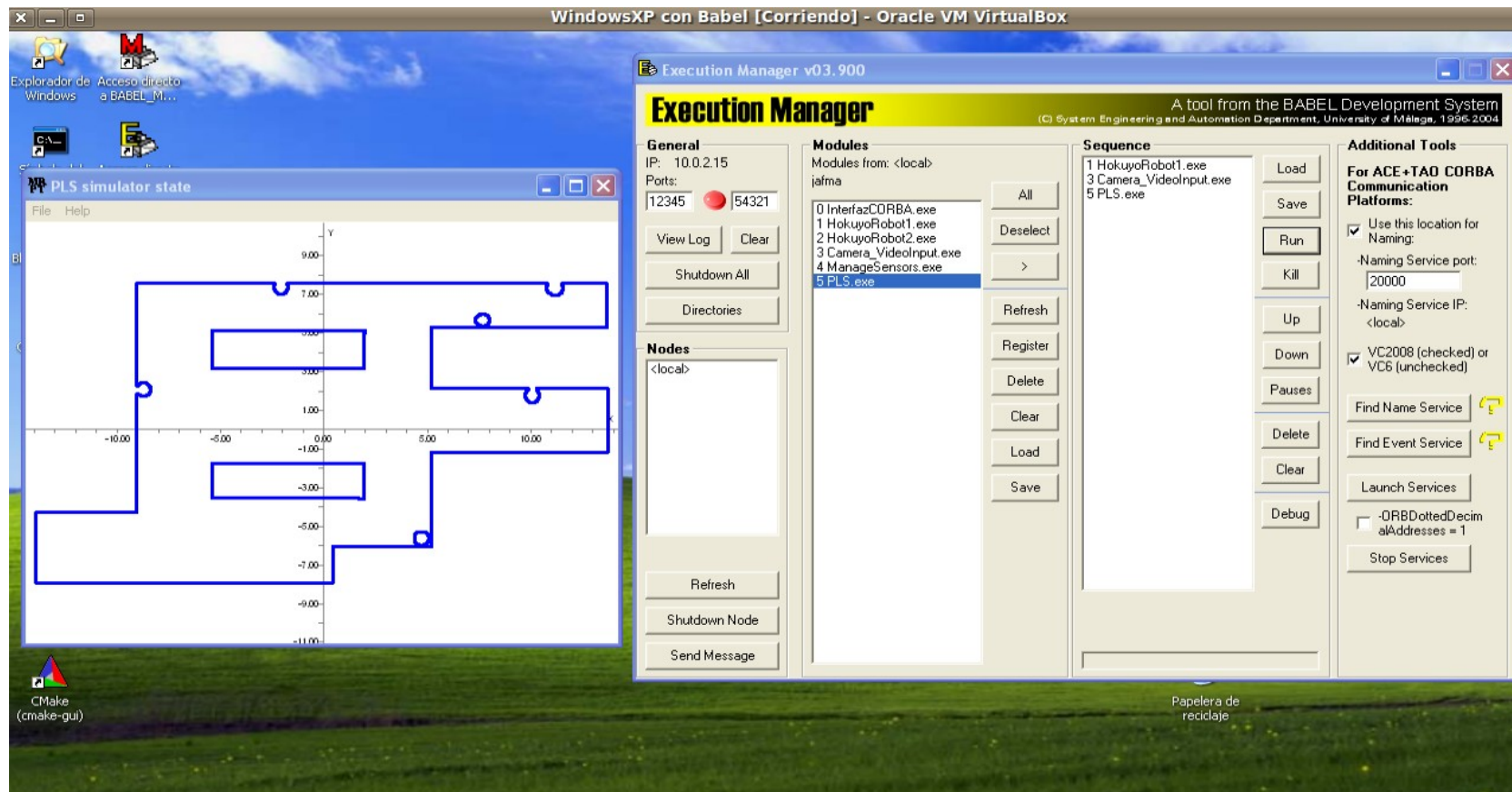
4.- Otros

BABEL:



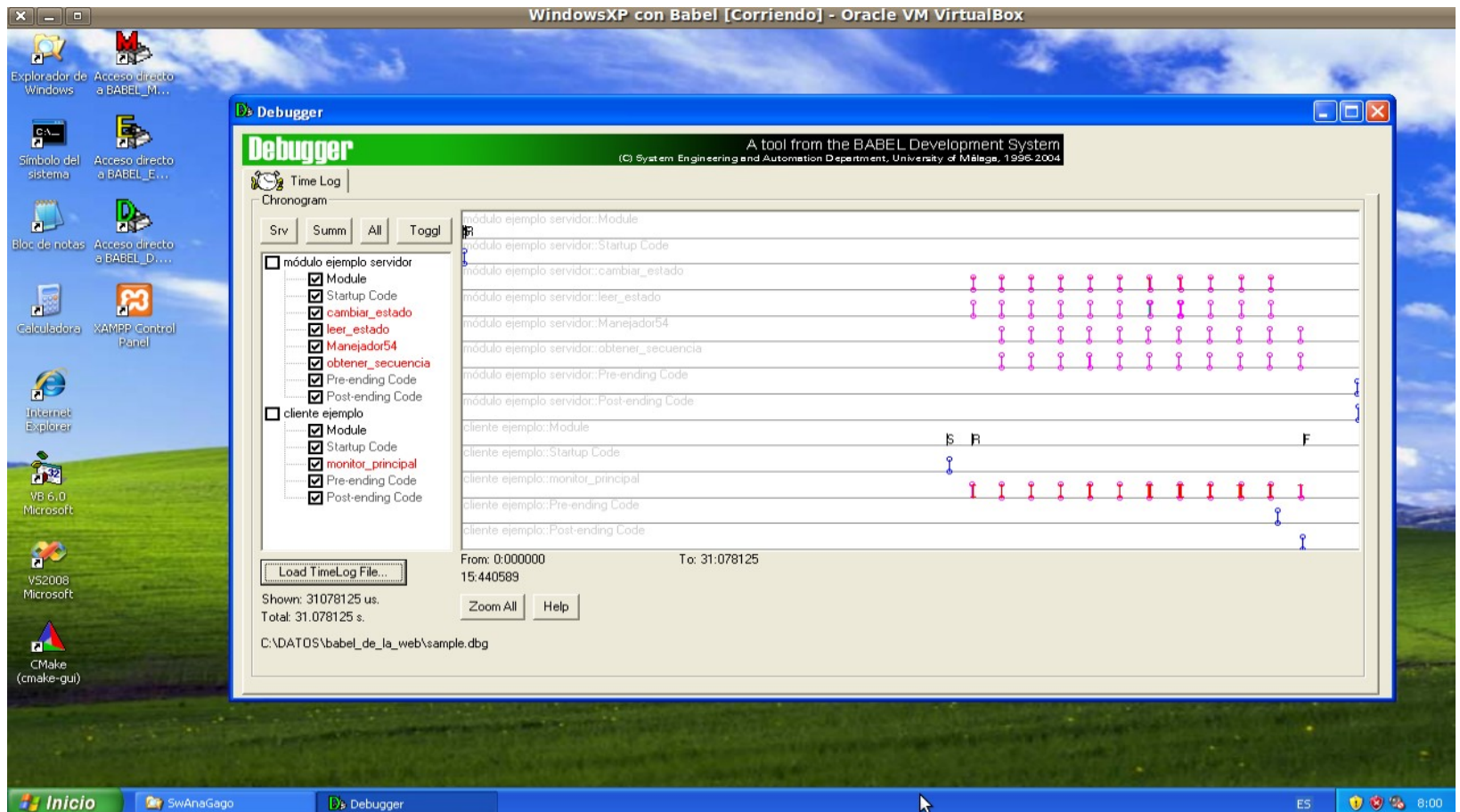
4.- Otros

BABEL:



4.- Otros

BABEL:



4.- Otros

OpenMora:

- Open Mobile Robot Architecture: arquitectura robótica basada en MOOS y MRPT
- <http://sourceforge.net/p/openmora/home/Home/>
- http://babel.isa.uma.es/babel2/index.php/Main_Page



4.- Otros

OpenMora:

- Proporciona módulos para plataformas robóticas comunes, para realizar tareas de localización, navegación, etc.
 - MOOS
 - Middleware para robots
 - <http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php>
 - Estado experimental
-

4.- Otros

V-Rep:

- <http://www.coppeliarobotics.com/index.html>
 - Simulación
 - Versión educativa
 - Robots manipuladores y robots móviles
 - Integrable con ROS
 - Código libre
-