UNIVERSIDAD PABLO DE OLAVIDE SEVILLA
ESCUELA POLITÉCNICA SUPERIOR

Robótica y Visión Artificial
Ingeniería Informática en Sistemas de Información

EPD 1: ROS introduction. Coordinate Frames with TF

Objectives

- First contact with ROS ROS (Robot Operating System).
- Understanding the ROS TF package for working with coordinate frames.

Cuestiones

1 Materials

In this and the remaining EPDs we will use the following tools:

- ROS (Robot Operating System): http://www.ros.org
    - In particular, we will use the ROS Kinetic distribution over Ubuntu 16.04 LTS (http://wiki.ros.org/kinetic/Installation/Ubuntu).

We will employ in the EPD some machines with Ubuntu 16.04 LTS and ROS Kinetic (ros-kinetic-desktop-full) installed.

To work at home, there are two possibilities:

- Installing ROS “natively”. This corresponds to implement all the following steps (in case you have your own computer, you have to complete all of them to follow the class. In this case, it is recommend to install the full desktop version of ROS):
    - http://wiki.ros.org/kinetic/Installation/Ubuntu
    - The same can be done even more easily the two-line installation script at http://wiki.ros.org/ROS/Installation/TwoLineInstall

2 ROS basic functionalities

2.1 Creating a ROS workspace

The first step is to create a ROS workspace. For that, you should follow Section 3 of http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

2.2 ROS Packages and stacks:

As commented, ROS is a collection of development facilities, libraries, communication layer and algorithms. All this is organized into packages. A description of this organization is given by the following tutorial:

http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem

There are several commands of interest. From this first tutorial, you should understand well the command **roscd.**

2.3 Creating a package:

A package is the main software organization unit in ROS. Each package can contain libraries, executables, scripts, etc. To learn how to create a package, we have the next tutorial:

http://wiki.ros.org/ROS/Tutorials/CreatingPackage

First, we will create a package called epd1. In the sandbox folder, create a package called epd1 that depends on tf and roscpp:

```
$ cd %YOUR_CATKIN_WORKSPACE_HOME%/src
```

```
 $ catkin_create_pkg epd1 tf roscpp
```

Build your new package before you can roscd:

```
 $ cd %YOUR_CATKIN_WORKSPACE_HOME%/
 $ catkin_make
 $ source ./devel/setup.bash
```

*Have a look into de new package by commanding **roscd epd1***

The two mandatory files of a package are the following:

- package.xml
- CMakeLists.txt

2.3.1 Package file

The package file is used to list the package name, version numbers, authors, maintainers, and **the dependencies of our code**. We can have internal dependencies to other ROS packages, and/or external dependencies to third-party libraries. This file is used by the ROS building system to install and/or compile these dependencies.

Open the file package.xml

First, the information about the name, etc, is indicated in the following tags:

```
<package>
  <name>foo_core</name>
  <version>1.2.4</version>
  <description>
  This package provides foo capability.
  </description>
  <maintainer email="ivana@willowgarage.com">Ivana Bildbotz</maintainer>
  <license>BSD</license>
</package>
```

Then, the dependencies are listed. Packages can have four types of dependencies:

- **Build Tool Dependencies** specify build system tools which this package needs to build itself. Typically the only build tool needed is **catkin**. They are indicated as:

```
<buildtool_depend>catkin</buildtool_depend>
```

- **Build Dependencies** specify which other packages are needed to build this package. This is the case when any file from these packages is required at build time.

```
<build_depend>”package name”</build_depend>
```

- **Run Dependencies** specify which packages are needed to run code in this package, or build libraries against this package.

```
<run_depend>”package name”</run_depend>
```

- **Test Dependencies** specify only *additional* dependencies for unit tests. They should never duplicate any dependencies already mentioned as build or run dependencies.

```
<test_depend>”package name”</test_depend>
```

2.3.2 CMakeLists file

In CMakeLists.txt, we indicate the files to compile and the libraries to link. ROS uses **catkin** (http://wiki.ros.org/catkin/conceptual_overview) as building system, which itself employs **CMake** (http://www.cmake.org). It is out of the scope of this course to explain in detail CMake. We will provide you with the adequate information when needed.

3 The TF package

The tf package is employed in ROS to deal with coordinate frames and their transformations. A description of the package can found at: http://wiki.ros.org/action/fullsearch/tf

This package provides tools that allow creating coordinate frames, composing transformations, etc.

The previous package epd1 does not have yet any code in it. We will create now a node (a process) in the package that creates and sends a coordinate frame transformation.

*Now, download the file called **c1.cpp** from AulaVirtual and copy it to the **src/** folder of your package.*

The code will be explained during the class.

*Now that we created the code, let's compile it first. Open the CMakeLists.txt file, and add the following line in the **build** space:*

```
add_executable(c1 src/c1.cpp)
```

With this line, we are indicating that we will add an executable called c1 to our package, and that this executable is made by the compilation of the file c1.cpp, located in the folder src.

We have to link our executable with the adequate libraries. This is done by adding the following line after the previous one:

```
target_link_libraries(c1 ${catkin_LIBRARIES})
```

In this example, only the libraries indicated here are needed.

Building a package means compiling all the source files and linking with all the required dependences. If the package.xml and CMakeLists.txt files are adequately filled, making a ROS package is done by the following command:

```
$ catkin_make
```

at the top folder of your catkin workspace

Build your package.

If everything went well, you should have a binary file called **c1** in your **devel/lib/epd1** folder.

In order to run the current code, we have the order:

```
rosrun <package> <executable> [params]
```

which it will run a given executable of a particular package.

Type the following command:

```
rosrun epd1 c1
```

NOTE: Remember that, to run a ROS node, you should have the ROS master node already running in your system. Type the following command in a different terminal:

```
roscore
```

To see the results, we have different tools:

- `tf_echo <source_frame> <target_frame>` Print information about a particular transformation between a source_frame and a target_frame. For example, to echo the transform between /world and /frame_1:

```
rosrun tf tf_echo /world /frame_1
```

- Run the visualization tool of ROS, rviz

```
rosrun rviz rviz
```

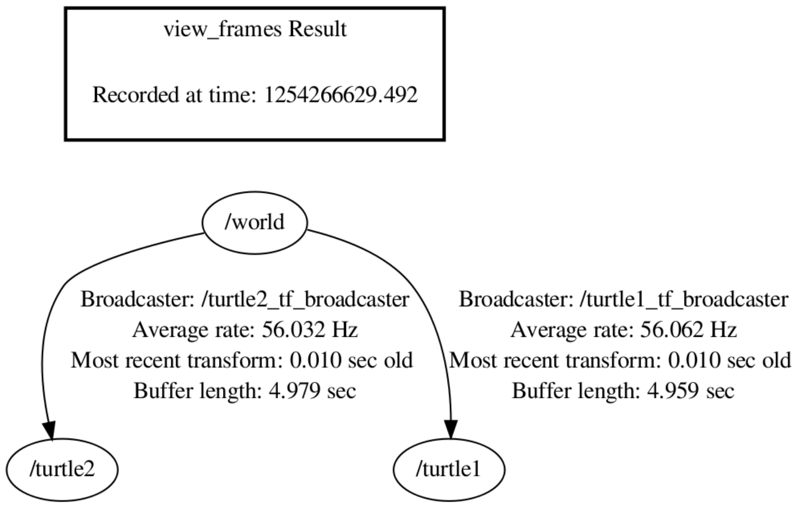
4 The TF tree

In a robot, the set of all transformations can be seen as a tree, in which each node of the tree is a coordinate frame, and the branches are the transformations between frames.

If you want to visualize the current tree of transformations, you can use the following command:

```
rosrun tf view_frames
```

The command should create a pdf file called frames.pdf, with a figure like the following one:

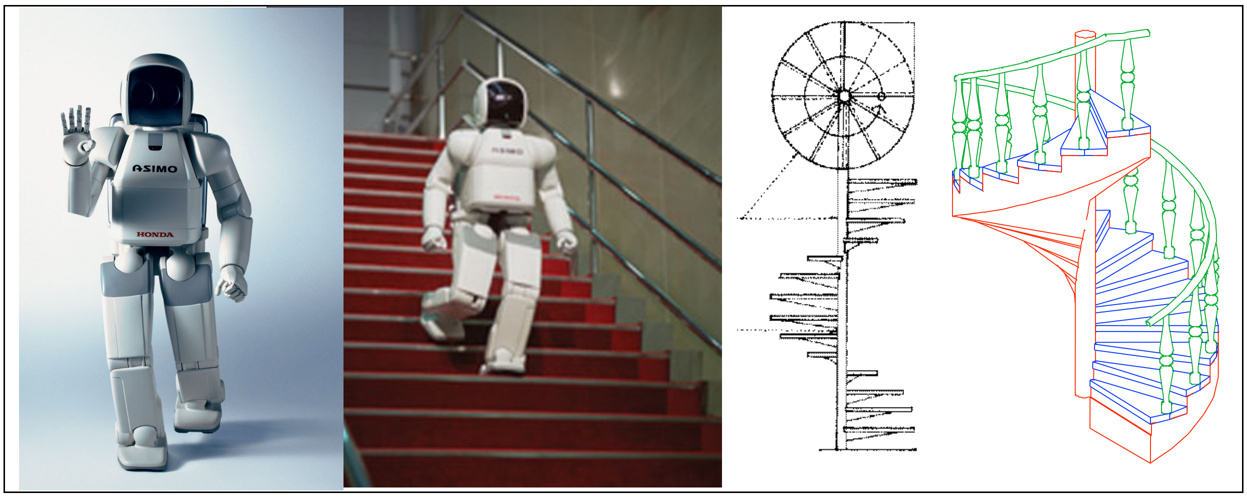

view_frames Result
Recorded at time: 1254266629.492
/world
Broadcaster: /turtle2_tf_broadcaster
Average rate: 56.032 Hz
Most recent transform: 0.010 sec old
Buffer length: 4.979 sec
Broadcaster: /turtle1_tf_broadcaster
Average rate: 56.062 Hz
Most recent transform: 0.010 sec old
Buffer length: 4.959 sec
/turtle2
/turtle1

5 Escalera de caracol.

Los ingenieros de Honda han construido el robot caminante ASIMO, capaz de subir y bajar escaleras rectas. Ahora quieren enseñarle a ASIMO a subir por una escalera de caracol. Para ayudarles en esta tarea, defina un sistema de referencia en cada escalón utilizando TF, y publíquelos todos (cada escalón tiene que tener un nombre distinto).

Visualice gráficamente los sistemas de referencia.

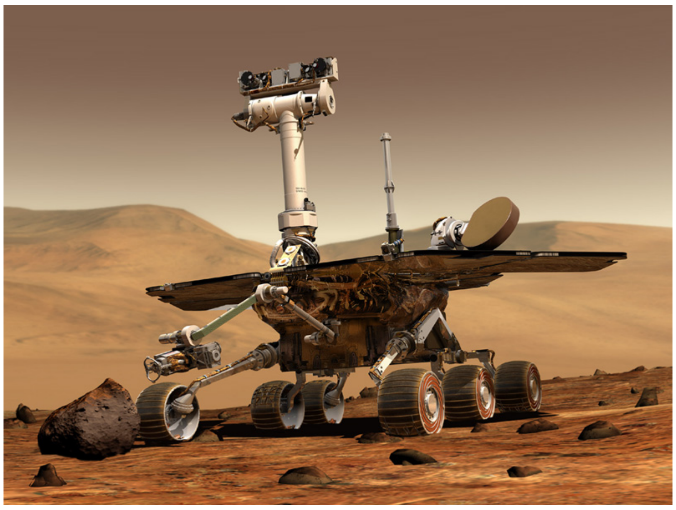
El origen del sistema de referencia estará situado en el borde del escalón, a una distancia del eje de la escalera de 2/3 del ancho total del escalón. La altura total de la escalera es de 2.70 m, y tiene 14 escalones de 1.2 m de ancho cada uno.



Para realizarlo, dispone del código c2.cpp. Añada ese código al paquete epd1, y analícelo. Puede partir del él.

6 Mars Rover: robot para exploración de Marte.

En el Jet Propulsion Laboratory de la NASA han diseñado un robot móvil de 6 ruedas par la exploración de la superficie de Marte. Se han construido dos robots, ***Spirit*** y ***Opportunity***, que han sido enviados a Marte para explorar su superficie y buscar indicios de la existencia de vida en ese planeta (presencia de agua). El Mars Rover ha sido diseñado para que pueda moverse por la superficie de Marte, sorteando rocas y otros accidentes del terreno. Para garantizar la seguridad del robot, el ángulo de roll (balanceo) debe estar entre +/- 15º y el ángulo de pitch (inclinación) entre +/- 22º.



Para ayudarles en sus análisis de fiabilidad, realice una simulación del movimiento del Mars Rover entre los dos casos límite. En la posición inicial los dos ángulos son nulos. Luego irá a la posición extrema roll = +15º y pitch = +22º. En la posición final roll = -15º y pitch = -22º.

Representar utilizando TFs el movimiento del Mars Rover entre estos puntos, asociando un sistema de referencia al robot. Se representarán 16 posiciones intermedias (21 en total). La distancia recorrida según el eje X entre cada dos posiciones será de 0.5 m, y el tiempo entre cada una de ellas de 1 segundo.

Visualice el resultado con rviz

Para realizar el problema, dispone como guía del código c3.cpp