

PROGRAMACIÓN DE ROBOTS INDUSTRIALES

Alumno: Hernandez Vidrio Victor Fabian.

Maestro: Moran Garabito Carlos Enrique.

Carrera: Ingeniería en Mecatrónica.

Matricula: 173112962.Grupo: 6°A.

Manipulacion de robot de modo manual

# Entornos de Simulación de Robots

A la hora de afrontar el diseño y construcción de un robot, se hace necesario disponer de mecanismos capaces de realizar una simulación lo más fiel posible de entornos reales. El elevado coste de la construcción puede ser tan alto que podría hacer inabordable la construcción de nuestro proyecto.

Gracias a que los ordenadores domésticos son cada vez más potentes, el uso de estos simuladores es hoy una realidad.

## Requisitos del software analizado

Actualmente existen un elevado conjunto de simuladores con mayor o menor calidad y capacidad.

### Morfología del robot

Un brazo robótico está formado por varios elementos como son la estructura mecánica, transmisiones, sistemas de accionamiento, sistemas sensoriales, sistemas de control y potencia, y elementos terminales. Además, mecánicamente el robot está compuesto de eslabones unidos mediante articulaciones. Todos estos elementos tienen una dimensiones y comportamientos físico que conforman el funcionamiento del robot.

Existen en el mercado transmisiones, reductores, actuadores y sensores con unas especificaciones de funcionamiento definidas de fábrica, y en este sentido los simuladores deberían disponer o bien de una base de datos con diferentes elementos existentes en el mercado o al menos la posibilidad de asignar valores o rangos a los elementos utilizados en nuestro robot modelo.

Otro punto a tener en cuenta a la hora de analizar un simulador, es la posibilidad de añadir o crear nuestro brazo robótico completo o utilizar uno existente en el mercado. Esta última opción es muy interesante ya que estaríamos en disposición de hacer pruebas o simulaciones con un sistema real.

En general, una de los requisitos que deberá tener un buen simulador es la capacidad de representar gráficamente y a ser posible en 3D el brazo robótico, y deberá mostrar en tiempo real su movimiento. Será recomendable disponer de la posibilidad de mover la cámara y realizar zoom sobre una zona en concreto, permitiendo de esta forma comprobar el funcionamiento completo del sistema.

Además de la representación del robot, sería recomendable que se permitiera añadir elementos y obstáculos en el entorno del robot que interactuaran con el mismo. De esta forma se podría realizar una simulación mucho más realista.

### Cinemática del robot

Una vez definido como será el robot físicamente, el simulador deberá permitir y controlar la cinemática propia del robot para evitar movimientos imposibles. El simulador deberá permitir mover tanto los eslabones como las articulaciones respetando la cadena cinemática. Además, se deberá respetar en todo momento las posibles restricciones de movimiento definidos en cada eje.

### Dinámica del robot

La Dinámica del robot relaciona el movimiento del robot y las fuerzas implicadas en el mimo. En este sentido, el simulador deberá permitir definir los valores propios a cada elemento del robot. Deberá tener en cuenta todas las fuerzas implicadas en el sistema y deberá actuar en consecuencia ante movimientos o cargas elevadas.

La morfología, cinemática y dinámica del robot ya han sido tratados en otras entradas anteriores. [Robótica: Cinemática y Dinámica del Robot](http://quepuedohacer.es/blog/ray/2016/06/30/robotica-cinematica-y-dinamica-del-robot/)

### Entorno

Como se ha indicado anteriormente, el sistema debería permitir definir el entorno en el que opera el robot y además debería disponer de la posibilidad de añadir sensores que respondan ante las condiciones indicadas en la simulación.

Para poder definir bien el entorno se debería permitir modificar:

* **Sistemas de física**: Por norma general la física aplicada en el brazo robótico estará definido entre otros por la fuerza de gravedad, pero sería interesante poder modificar estos valores para cambiar el entorno de trabajo a situaciones más extremas. Por ejemplo, se podría eliminar la gravedad para comprobar el funcionamiento en entornos ingrávidos como por ejemplo el espacio, o aumentar el nivel de atmósfera para simular situaciones submarinas. Otros valores que se podrían modificar serían los valores de humedad y temperatura (por ejemplo) que deberían afectar al movimiento de los motores y actuadores.
* **Sistemas de renderizado 3d**: El sistema de renderizado 3D es otro punto a tener en cuenta y sería interesante que el renderizado sea lo más realista posible sin que por ello el sistema se vea ralentizado.
* **Sistemas de modelado**: Este es un detalle menor, pero la existencia de un sistema de modelado dentro del propio simulado facilitaría la construcción y modificación del robot. Si pensamos en que estamos creando un robot, es posible que para ciertos movimientos las propias piezas choquen entre sí, y si el simulador permitiese realizar el ajuste en la propia interface ganaríamos agilidad. Al menos el simulador debería admitir modelos 3D en los formatos estándar.
* **Entorno de diseñado y programación**: Otro punto a tener en cuenta a la hora de analizar el simulador es si la ejecución del mismo se realiza desde un GUI (Graphical User Interface – Interfaz gráfica de usurio) o por el contrario la simulación se programa completamente con scripts y luego se lanza la simulación por línea de comandos.

### Programación

En este punto diferenciaré dos tipos de programaciones:

* **Sistema**: Un buen simulador debería permitir la creación de nuevos módulos para implementar aquellas situaciones o comportamientos no contemplados en el propio simulador. El sistema debería disponer de un API lo suficientemente amplio para permitir la programación de estos añadidos.
* **Robot**: No podemos olvidar que uno de los objetivos del simulador es el de llevar a la realidad las pruebas realizadas. La mayoría de los robots modernos son programables y por tanto disponen de la capacidad de cargar ficheros con el código a ejecutar. El simulador debería permitir realizar y exportar el movimiento empleado en la simulación en un lenguaje comprensible por el robot real, o al menos a un lenguaje que permita su conversión sobre un software especializado.

### Conexión

Aunque se trate de un simulador, es importante que el sistema simulado funcione como un sistema real. En este sentido, el robot debería trabajar de forma autónoma, comunicándose con el sistema de control mediante alguno de los protocolos más habituales. En este sentido se verán que tipos de protocolos se utiliza en cada simulador.

# Simuladores

A la hora de analizar los simuladores sol dividiremos en 2 tipos en función de si su uso requiere una licencia o por el contrario son de uso libre.



### Gazebo

### Gazebo es un software de simulación y su principal ventaja, además de disponer de una licencia Apache 2 (y por tanto libre), es que está especialmente diseñado para probar de forma rápida algoritmos y diseño de robots.

<http://gazebosim.org>

#### Morfología del robot

Para la parte de renderizado, Gazebo cuenta con la ayuda de [OGRE](http://www.ogre3d.org/about/features). Este software ofrece un entorno sencillo para el diseño orientado a objetos e independiente de la implementación 3D, por lo que utiliza tanto DirectX como OpenGL.

Gazebo dispone de un editor que permite añadir figuras básicas como cilindros, esferas o cubos, u otras más elaboradas basadas en gráficos SVG extruibles o mayas 3D en formato “.dae” o “.stl”. Para crear este tipo de mayas se puede utilizar software libre como por ejemplo [Blender](http://www.blender.org).

Para unir las figuras y conformar el robot, dispones de uniones en las que se puede definir el tipo de movimiento permitido (Rotación, Prismático, esférico, etc). Además, permite definir los límites de movimiento, límites de fuerza soportada y velocidad soportada, viscosidad, fricción, etc.

Otra posibilidad que ofrece Gazebo es la de importar robots mediante el formato SDF. Este formato basado en XML describe objetos y entornos para simulación de robots, visualización y control. Además de permitir desarrollar nuestros propios prototipos, el sistema dispone de modelos propios.

#### Dinámica y Cinemática

Como se ha visto en el apartado anterior, es posible la parametrización de los diferentes elementos que intervienen en el robot. Además, Gazebo dispone de un conjunto amplio de clases matemáticas que permite definir y operar por ejemplo con matrices, quaternios, planos, etc.

En cuanto a la dinámica, es compatible con varios motores físicos como:

* [ODE](http://www.ode.org/) (Open Dynamics Engine) que es una librería para la simulación de cuerpos rígidos y permite el control de colisiones y rozamientos.
* [SIMBODY](http://simtk.org/home/simbody/). Esta librería dota a nuestros objetos de capacidades dinámicas, resolviendo por ejemplo la segunda ley de Newton (F=m\*a) (Fuerza es igual a la mása por la aceleración).
* [DART](http://dartsim.github.io/) (Dynamyc Animation and Robotics Toolkit). Esta librería ofrece estructuras de datos y algoritmos para cinemática y dinámica.
* [BULLET](http://bulletphysics.org/wordpress/). Esta librería de Código Abierto es un motor de detección 3D y dinámica de cuerpos tanto rígidos como blandos.

#### Entorno

Gazebo dispone de una interface en el que se permite diseñar todo el entorno en el que operará el robot, dotando a todos los objetos las características de cuerpos rígidos con colisión. Además de colisiones propias de los objetos, se dispone de la posibilidad de añadir sensores de tipo laser, cámaras 2D o 3D, sensores de fuerza, de contacto, etc. Todos estos sensores pueden ser incluidos con ruido, con lo que se consigue una simulación más real.

Otro punto a favor, es la posibilidad de realizar simulaciones remotas y en la nube.

#### Programación

En cuanto a la programación del sistema, y al tratarse de un software de código abierto, se dispone de la posibilidad de desarrollar plugins específicos que alteren el comportamiento del robot, de los sensores o del entorno, posibilitando la configuración de cualquier situación.

Para ello dispone de un API muy completo de programación en C++.

En cuanto a la generación de programas para implantarlos directamente en robots, no parece que exista directamente esta posibilidad.

#### Instalación y pruebas de simulación

Aunque el sistema a partir de la versión 6.0 es compatible con Windows, no he conseguido (aún) instalarlo correctamente en este Sistema Operativo. Sin embargo, la instalación en una máquina con Linux (concretamente en una distribución Ubuntu) es sencilla y automática.

### 

### MORSE

Es un simulador académico y está orientado a la simulación 3D en grandes entornos.

<https://www.openrobots.org/wiki/morse>

#### Morfología del robot

MORSE utiliza Blender como motor gráfico y de “virtualización del entorno” y gracias al generador de juegos que incluye el propio Blender es capaz de simular la física de forma adecuada.

Dispone de varios dispositivos, sensores y actuadores con los que podemos trabajar, pero es capaz de cargar cualquier objeto 3D en varios formatos, lo que nos ofrece la posibilidad de crear cualquier tipo de robot.

#### Dinámica y Cinemática

No parece que disponga de un sistema dinámico y cinemático propio, recayendo en el programador del sistema a simular la labor de desarrollar toda esta tarea. Aun así, Blender dispone de sistemas de colisión y algunas físicas básicas que podremos utilizar en nuestra simulación.

#### Entorno

No se disponen de un entorno, al uso, más allá de la simulación realizada por el propio Blender o la utilización de este para el desarrollo de los objetos utilizados en la propia simulación.

#### Programación

Se utilizan scripts en Python para definir todo el entorno de simulación y el comportamiento de cada elemento incluido en la escena. Aunque dispone de una librería de clases básicas, es más o menos sencillo ampliarla para adaptarla a nuestras necesidades.

#### Instalación y pruebas de simulación

El software fue concebido para correr sobre Linux, y la instalación parte por compilar el código fuente enlazándolo con el Blender.

En la prueba realizada, aparece una incompatibilidad entre la versión de Python utilizada en la compilación del Blender máximo admitido por MORSE. Para solucionarlo he tenido que compilar ambos softwares (Blender y MORSE) con la misma versión de Python para hacer que funcionase.

### 

### SimRobot

Este software de simulación de robot fue desarrollado por la [Universidad de Bremen](http://www.uni-bremen.de/) y el [Centro Alemán de Investigación en Inteligencia Artificial](http://www.dfki.de/) (DFKI).

<http://www.informatik.uni-bremen.de/simrobot/index_e.htm>

#### Morfología del robot

Este simulador utiliza [GLEW](https://sourceforge.net/projects/glew/) (The OpenGL Extension Wrangler Library) para generar los gráficos 3D, y para definir el robot se utiliza el Lenguaje de Marcado [RoSiML](http://www.informatik.uni-bremen.de/spprobocup/RoSiML.html) (Robot Simulation Markup Language), el cual permite diseñarlo a partir de objetos básicos.

#### Dinámica y Cinemática

La Cinemática se define en base a uniones y atributos físicos como la Fricción, la Masa y el centro de Masa de cada parte del robot.

<PhysicalAttributes>

<Mass value=" xs:double [1]"/> [0..1] ?

<CenterOfMass> Triple </CenterOfMass> [0..1] ?

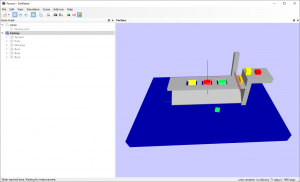
<Friction direction1=" xs:double [1]" direction2=" xs:double [0..1]"/> [0..1] ?

</PhysicalAttributes>

Para la Dinámica utiliza ODE.

#### Entorno

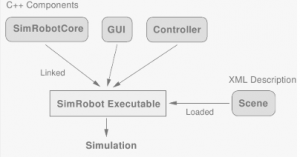
No dispone de un Interface para la construcción del entorno, pero sí dispone de una interface para la ejecución de la simulación.



#### Programación

El proyecto está desarrollado en C++ y utiliza Qt para hacerlo portable a la mayoría de plataformas.

A la hora de diseñar una nueva simulación se ha de construir un controlador que incluya todos los sensores y disponibles en la escena. El esquema básico del sistema es:



#### Instalación y pruebas de simulación

No dispone de un instalador como tal, sino que se dispone del código del proyecto compilable.

### v-rep

Este simulador desarrollado por Coppelia Robotics, permite modelar un entorno completo. Dispone de dos modalidades de licencia:

* Educacional: Esta licencia nos permite hacer uso completo del software.
* Comercial: También se dispone de una versión de evaluación que no permite el guardado y una versión de visualización.

También se dispone del código del software.

[www.coppeliarobotics.com](http://www.coppeliarobotics.com)

#### Morfología del robot

Dispone de un simulador 3D propio y que es capaz tanto de editar los modelos como de importar y exportar modelos desde un amplio número de formatos. Además de la capacidad para importar modelos, el sistema dispone de un conjunto de modelos básicos que nos permitirán realizar las simulaciones deseadas.

#### Dinámica y Cinemática

Dispone de 4 sistemas de dinámicas:

* Bullet
* ODE
* [Newton](http://newtondynamics.com/forum/newton.php). Esta librería permite dotar de física a nuestros objetos.
* [Vortex](http://www.cm-labs.com/). Esta librería proporciona la posibilidad de dotar a nuestro entorno de valores realistas y precisos. V-rep lo utiliza, entre otras cosas, para calcular la distancia entre los diferentes objetos de la escena.

Esto nos permite dotar a nuestro sistema de las físicas necesarias para simular un entorno real.

Para la cinemática dispone de sistemas de cálculo tanto para la cinemática inversa (IK) como al directa (FK).

#### Entorno

El software dispone de una interface integrada que nos permite diseñar todo el entorno y añadir nuestros modelos con la simple acción de coger y soltar.

En cuanto al entorno, el simulador dispone de:

* Un sistema de colisión.
* Utiliza [OMPL](http://ompl.kavrakilab.org/) (The Open Motion Planning Library) como algoritmo de planificación de movimiento basadas en rutas.
* Dispone de sensores de proximidad, visión, etc.
* Dispone de un sistema de partículas y fluidos.

Otra de las posibilidades que ofrece este software es la posibilidad de definir diferentes entornos, dotando a nuestra simulación de la capacidad de trabajar en fluidos o en cualquier otra situación.

#### Programación

V-REP dispone de un API muy completo, disponiendo de varias formas de trabajar:

* API normal: Dispone de más de 500 funciones programadas en c++ y [Lua](http://www.lua.org) desde las que se pueden controlar la escena y los modelos.
* API externas: Dispone de cerca de 100 funciones que permiten enlazar nuestra simulación con otras librerías externas. Estas se pueden programar en (C/C++, Phyton, Java, Matlab, Octave y Lua).
* Interface [ROS](http://www.ros.org) (Robot Operating System). Este framework dispone de herramientas, librerías y algunas convenciones que ayudan a simplificar la creación de robots complejos y robustos.

### Microsoft Robotics Studio Visual Simulation Environment (MRDS4)

MRDS es un simulador desarrollado por Microsoft, y permite la creación y simulación de nuestros propios robots. Este software ofrece varios métodos y tecnologías para la creación rápida de prototipos e incluye un gran número de librerías.

<https://www.microsoft.com/en-us/download/details.aspx?id=29081>

Actualmente este software ya no se actualiza desde el 2014 lo que reduce la información disponible en Internet sobre él.

#### Características

El simulador solo funciona bajo Windows y aunque dispone de un entorno gráfico aceptable, no dispone de la posibilidad de desarrollar robots de forma sencilla, aunque incorpora de serie una serie de modelos que permitirían utilizarlos para probar algoritmos.

Es un entorno cerrado en el sentido de que no dispone del código de propio simulador, ofreciendo únicamente una librería de uso y comunicación con los diferentes elementos. Toda la programación se realizará con C# o VPL (<https://msdn.microsoft.com/en-us/library/bb483088.aspx>).

### RobotStudio

RoboStudio es un simulador desarrollado por la empresa ABB. Aunque es un software que requiere licencia, se dispone de una versión de evaluación por 30 días.

<http://new.abb.com/products/robotics/robotstudio>

#### Morfología del robot

Dispone de un gran conjunto de brazos robóticos propios y un sistema de renderización muy conseguido, lo que es tanto una ventaja como un inconveniente. Por un lado, al utilizar sus propios robots, el comportamiento de los mismos dentro del entorno simulado es casi idéntico al entorno real lo que dota a la simulación de efectividad. Por el contrario, no se dispone de la posibilidad de realizar pruebas con elementos ajenos a los ofrecidos por el fabricante y que no estén previamente cargados en el sistema.

#### Dinámica y Cinemática

Al ser un sistema cerrado, el comportamiento de la cinemática del robot y de la Dinámica del sistema es muy similar al esperado en la vida real para los robots existentes.

#### Entorno

Dispone de una interface gráfica que facilita la ayuda a la hora de definir el entorno de simulación.

Respecto a esto, una de los mayores inconvenientes detectados, es que no siempre es sencillo moverse en el espacio 3D. No utiliza el sistema de navegación común a herramientas de diseño 3D y acciones como rotación y zoom no es sencillo.

#### Programación

Una de las ventajas de este simulador, es la posibilidad de hacer programas en Rapid para que sean ejecutadas por nuestro robot. Esto ayuda a exportar nuestras pruebas a robots reales.

#### Opinión

Es un entorno muy amigable y sencillo de manejar. Una de las grandes ventajas es que dispone de la posibilidad de comunicar directamente nuestro robot simulado con un robot real, consiguiendo replicar la posición entre ambos.

El inconveniente es que no dispone de la posibilidad de probar otros robots que no sean del fabricante ABB, al menos de una forma sencilla.

### RoboDK

Es un simulador industrial con el cual se pueden simular sobre 200 robots y herramientas de distintos fabricantes como ABB, KUKA, Yaskawa, etc.

Dispone de una versión libre, aunque con limitaciones.

<http://www.robodk.com>

#### Morfología del robot

El simulador dispone de un número considerable de robots y herramientas de diferentes fabricantes que permiten una rápida puesta en marcha. Lo único que deberemos hacer es seleccionar el tipo de robot o herramienta y arrastrarla a la zona de diseño.

También dispone de la posibilidad de importar nuestros propios modelos 3D.

#### Dinámica y Cinemática

Hay poca información al respecto sobre los sistemas de dinámica y cinemática que utiliza este simulador, pero las pruebas que se han realizado hacen pensar que no dispone de un sistema de física.

En cuanto a la cinemática parece que está resuelta al menos en la Inversa, permitiendo manejar cualquier eslabón o articulación y propagando el movimiento a sus uniones de forma correcta y respetando todas las restricciones.

#### Entorno

El entorno es muy amigable, permitiendo la incorporación de brazos robóticos y herramientas de forma sencilla y rápida. Quizás se echa un poco en falta la posibilidad de añadir elementos al entorno que permitan dotar a la simulación de mayor realismo.

#### Programación

La programación es uno de los puntos fuertes de este software ya que permite la creación de scripts en Python que modifiquen el comportamiento del brazo robótico, realizando los movimientos necesarios para el trabajo a realizar, y disponiendo de la posibilidad de generar el código en RAPID el cual puede ser cargado en robots físicos que realizarán el trabajo planificado.

En la versión trial, solo se generarán las 50 primeras instrucciones.

#### Opinión

Este simulador es muy adecuado cuando se necesitan realizar ajustes o implementaciones de movimiento en robots físicos.

Aunque no dispone de un sistema de física o dinámica parametrizable, la posibilidad de programar el entorno en Python nos permite suplir esta limitación, dándonos la alternativa de procesar nuestros cálculos en herramientas externas como Matlab.

Este es un simulador muy adecuado cuando disponemos de uno o varios brazos robóticos a los que queremos dotar de un movimiento concreto.

### Simuladores de Visual Components

Este simulador de Visual Components permite la visualización de una línea de producción en un entorno industrial. Está diseñado para la representación de todos los elementos que participan en la cadena de producción y está dividida en los siguientes softwares.

<http://www.visualcomponents.com/>

* 3DAutomate: Simula la automoción de una fábrica.
* 3DCreate: Permite la importación de modelos CAD y la simulación de líneas de producción.
* 3DRealiza y 3DRealize R: Ofrece la visualización 3D del modelado y la visualización de comportamiento de los equipos.
* 3DSilumate: Aporta la API para la presentación personalizada de la simulación.

#### Morfología del robot

Mediante 3DCreate, se puede construir nuestro modelo de robot mediante una interface gráfica sencilla. Desde esta interface se podrá parametrizar cada eslabón del robot o modelo que incluyamos en la simulación. Además, se incluye una librería de componentes entre los que se encuentran robots de fabricantes como ABB, Fanuc, KUKA, Mitsubishi, Panasonic, etc.

#### Dinámica y Cinemática

El Software resuelve tanto el problema de cinemática inversa como directa, permitiendo por ejemplo definir una serie de puntos por las que debería pasar la herramienta del robot.

Se desconoce si el simulador dispone de la posibilidad de modificar el comportamiento dinámico del robot o si dispone de físicas.

#### Programación

La programación se realiza desde Python mediante la utilización de una API.

### Actin Simulation

Simulador de Energid que permite la simulación de robots complejos.

<http://www.energid.com/software/actin-simulation/>

#### Morfología del robot

Permite la creación de nuestros modelos mediante el formato CAD.

#### Dinámica y Cinemática

El software es capaz de realizar el cálculo de la cinemática inversa en tiempo real con más de 100 grados de libertad.

Implementa cálculos de Newton-Euler para cuerpos rígidos usando algoritmos de composición de inercia de cuerpos rígidos y la inercia de articulaciones de cuerpos rígidos.

#### Entorno

Dispone de un Interface gráfico 3D que permite la manipulación del entorno. También incluye sensores y cámaras que permiten dotar a la simulación de comportamientos similares al mundo real.

#### Programación

La programación se realiza mediante ficheros de configuración usando XML los cuales tendrán acceso tanto a los modelos como a las herramientas matemáticas, permitiendo definir datos en diferentes sistemas como quaternios, ángulos de Euler, etc.

### Workspace

Es un entorno de simulación 3D que soporta una larga lista de lenguajes de programación utilizados en la industria de fabricantes de robots como ABB G-Code, ABB Rapid, Adept V-Plus, Fanuc Karel 5, Fanuc TP, Mitsubishi PA10, Mitsubishi Melfa Basic, Motoman Inform II, Kawasaki AS, Kuka KRL, Nachi Slim, Panasonic Pres y Siemens G-Code.

<http://www.workspacelt.com/>

Dispone de dos versiones:

* Workspace LT: De uso educacional
* Workspace: De uso industrial

#### Morfología del robot

Es compatible con CAD lo que permite la creación de nuestros propios modelos.

El software incorpora diferentes robots para su simulación.

#### Dinámica y Cinemática

Dispone de un sistema de colisión y de análisis dinámico y estático.

#### Programación

La programación de los robots se puede realizar mediante el uso del lenguaje nativo del robot ya que implementa los siguientes lenguajes:

* ABB – Arla, Rapid
* Adept – VPlus
* Esched – ACL
* Fanuc – Karel, TP
* JIRA – Strolic
* Kawasaki – AS
* Mitsubishi – MMB
* Motoman – Inform1, Inform2
* Nachi – Slim
* Unimation – Val1, Val2

### Webots

Es un entorno de simulación de robots creado por Cyberbotics.

<http://www.cyberbotics.com>

#### Morfología del robot

Aunque el software dispone de diferentes robots ya prediseñados, dispone de la capacidad de diseñar nuestros propios robots y herramientas mediante herramientas externas como Blender, 3D Max, Skechup, etc.

#### Dinámica y Cinemática

Para el cálculo y parametrización de la dinámica de nuestro sistema se utiliza ODE, incluyendo solo en la versión “Pro” la posibilidad de definir comportamientos avanzados como dinámicas de fluidos, sistemas sin fricción, etc.

#### Entorno

Dispone de una interface gráfica muy completa y un sistema de renderizado que permite la acción de coger y soltar. Todo apunta a que el sistema de renderizado es OGRE.

Se dispone de la posibilidad de añadir luces, nieblas, sombras y diversas texturas y transparencias que ofrecen la posibilidad de realizar simulaciones más realistas. Además, y como se ha indicado anteriormente, desde la interface se permite la edición de primitivas que pueden ser extruidas y por tanto modeladas. Esto junto con el sistema de colisiones y físicas dan una gran potencia al sistema de simulación.

El simulador dispone de una amplia librería de sensores que darán a nuestros robots la capacidad de medir distancias a otros objetos, obtención de imágenes, posicionamiento mediante GPS, brújula, acelerómetro, sensores de fuerza y presión, etc.

Otro elemento a destacar es la posibilidad de crear motores, permitiendo elegir su velocidad, aceleración, fuerza y otros parámetros.

#### Programación

En cuanto a las posibilidades de programación, se dispone de una API con cerca de 200 funciones disponibles en varios lenguajes de programación. Estas darán acceso a otros sistemas como OpenCV para el procesado de imágenes o Matlab para cálculos complejos.

A la hora de realizar nuestra simulación, es necesario especificar un controlador para cada uno de los elementos que deseamos manejar, el cual puede estar escrito en C/C++, Java, Python o MATLAB.

#### 3D Robot Simulation Tool

La herramienta de simulación de robots 3D está desarrollada por la empresa alemana EASY-ROB.

<http://www.easy-rob.com>

#### Morfología del robot

Dispone de una librería estándar con robots de adept / b+m / Denso / Eisenmann VarioRobots / Güdel / igm / Kawasaki / Manz-Automation / Mitsubishi / OTC-Daihen / Unimatio.

Opcionalmente dispone de la posibilidad de adquirir librerías para robots de ABB / FANUC / KUKA / MOTOMAN / PKM DELTA / STAUBLI / TRICEPT.

Incorpora un sistema de modelado 3D para primitivas (cilindros, conos, esferas y cubos) y es capaz de importar y exportar modelos complejos en los formatos más comunes (STL, 3DS, IGP-Format, VRML).

#### Programación

Dispone de un lenguaje de programación propio para los robots ERPL (Easy-Rob Program Language) aunque dispone de un API para integrar nuestros propios algoritmos en C para cinemática inversa, planificación e interpolación de movimientos, control dinámico y desarrollo de diálogos de usuario e interface de sensores.

Dispone de la posibilidad de generar el movimiento del Robot a uno de los siguientes lenguajes de fabricantes de robots:

* ABB
* KUKA
* BM
* OTC
* Fanuc
* Comau

También se dispone de la posibilidad de añadir nuevos lenguajes mediante un API de post-procesado.

### Marilou Robotics Studio

Este entorno de simulación de la empresa anyKode está diseñado tanto para robots humanoides como para brazos articulados o sistemas de robots paralelos en condiciones reales y respetando las leyes físicas. Dispone de una versión de evaluación por 30 días.

<http://www.anykode.com>

#### Morfología del robot

Incorpora una librería con motores, servos, sensores, cámaras, robots y mundos virtuales de prueba. Dispone de la capacidad de importar modelos en formatos como 3DS, X, OBJ, MESH o STL y es capaz de importar escenas de otros simuladores como Collada, Gazebo y URDF

#### Dinámica y Cinemática

En cuanto a la dinámica del robot, el simulador permite el ensamblado de cuerpos rígidos que dispondrán de masa e inercia. También es posible dotar a estos cuerpos de cierta flexibilidad para dotar a la simulación de mayor realismo.

Estos cuerpos rígidos estarán sujetos a factores como la gravedad, colisión u otros aspectos físicos.

Mediante la unión de los cuerpos rígidos se podrá definir el número de grados de libertad del sistema el cual dispone de cálculos para la cinemática inversa.

#### Entorno

Desde la interface que incorpora el simulador, es sencillo la inclusión tanto de robots como de otros elementos participantes en la simulación. Además, la división en 4 vistas (superior, frente, derecha y libre) permite una buena gestión de toda la escena.

En cuanto a la gestión del entorno, el sistema permite modificar de forma rápida aspectos como la fuerza de gravedad, velocidad de ejecución, viscosidad, etc.

#### Programación

La programación del robot se realiza mediante la SDK MODA (Marilou Open Devices Access) la cual dispone de librerías válidas para los lenguajes C/C++, C++ CLI, C#, J#, VB#.

Mediante esta SDK podremos enlazar nuestra aplicación con el robot simulado y acceder a sus sensores y actuadores para realizar la simulación deseada.

## Otros simuladores

Los simuladores vistos en los apartados anteriores han sido aquellos que más se ajustaban a los requisitos perseguidos. No obstante existen muchos más simuladores como:

* **OpenHRP**: Este entorno pretende simular el comportamiento físico y dinámico real.
  + <http://fkanehiro.github.io/openhrp3-doc/en/about.html>
* **Encanaçao**: Es un proyecto desarrollado en 1999 y del cual se dispone de código fuente.
  + <http://www.encarnacao.com/e_index.htm>
* **AUV Workbench:** Este software gratuito está diseñado para simular vehículos autónomos.
* **Sinbad:** Es un simulador desarrollado en Java. Utiliza Java 3D para la representación del mundo y está especialmente diseñado para vehículos, disponiendo de sensores de visión, sónicos, infrarrojos y pulsadores de contacto. El desarrollo del software está detenido desde el 2011.
  + <http://simbad.sourceforge.net/>
* **Robologix**: Es un simulado industrial diseñado principalmente como herramienta educacional.
  + <https://www.logicdesign.com/robotics_simulation.php>
* **WorkcellSimulator**: Simulador con entorno 3D para la programación de flujo de trabajos y máquinas con varios grados de libertad.
  + <http://www.it-robotics.it/products/3d-simulation/workcellsimulator/?lang=en>
* **SimSpark:** Es un simulador genérico de físicas con un sistema de visualización 3D, y dispone de un framework flexible.
  + <http://simspark.sourceforge.net/wiki/index.php/Main_Page>
* **Toolbox de robótica de MATLAB:** Herramienta disponible en Matlab para el cálculo de diferentes aspectos propios de un brazo robótico.

Como se ha visto a lo largo del post, actualmente se dispone de un gran número de simuladores disponibles para su utilización. La mayoría de ellos disponen de la capacidad de simular todos los aspectos físicos que afectan a un robot real lo que permitiría realizar simulaciones muy reales.

De entre los simuladores con licencia, y aunque “Webots” y “3D Robot Simulation Tool” parecen muy completos, el simulador “Marilou” destaca sobre ellos. Si bien es cierto que no dispone de la capacidad de generar código en el lenguaje propio del robot a simular, tal y como hace “3D Robot Simulation Tool”, la posibilidad de conectar el sistema a un código compilado previamente y que se disponga de una versión de pruebas totalmente funcional me parece muy interesante y por esta razón la he situado como una de las mejores opciones (En este criterio no se ha valorado ni el coste de la herramienta ni el soporte técnico o documentación existente).

Por otro lado, la existencia de librerías para el control de gráficos 3D como OGRE o Blender, la dinámica de cuerpos rígidos y físicas en general como ODE, ha conseguido que los simuladores con licencias de código libre se hayan vuelto muy potentes. Pero de entre los sistemas de software libre disponibles Gazebo es el que parece más completo.

En general la simulación de robots en la actualidad está totalmente conseguida y es asequible para cualquier usuario.

# Robótica: Cinemática y Dinámica del Robot

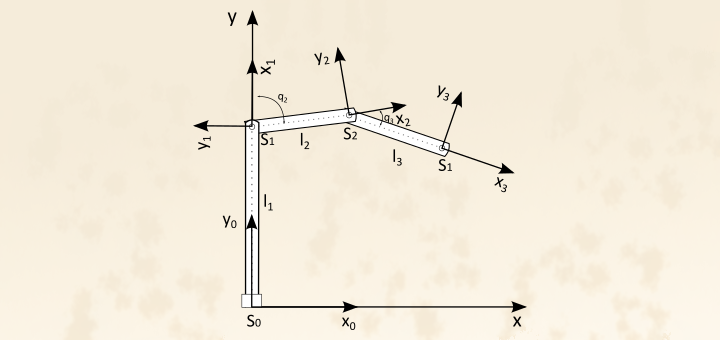
Se expondrá dos conceptos fundamentales que deberemos tener en cuenta en la construcción de nuestro brazo robótico. Estos concentos son la cinemática y la dinámica.

En esta ocasión, el dato es divulgativo y por tanto no me centraré en aspectos formales y en fórmulas matemáticas, sin embargo en 7mo, entraremos a detallar estos temas con un mayor nivel de complejidad.

## Estructura de un robot

Antes de empezar a definir el movimiento de un brazo robótico, deberemos ver cómo está estructurado el robot.

Podemos definir un brazo robótico como una cadena de elementos denominados eslabones que se unen mediante articulaciones.



Estas articulaciones darán movimiento a nuestro brazo y se dividen en:

* Movimientos prismáticos: Son movimientos de desplazamiento y típicamente en una única dimensión (hacia delante o atrás, arriba o abajo, izquierda o derecha).
* Movimientos rotacionales: Son movimientos de giro en una de las dimensiones (eje x, y o z).

Además de estos movimientos básicos podemos encontrar movimientos más complejos como los movimientos en esfera o “rótula” los cuales permiten una rotación en las tres dimensiones, movimientos planares que permiten el movimiento en dos dimensiones a la vez o movimientos cilíndricos que permiten un movimiento rotacional y prismático a la vez.

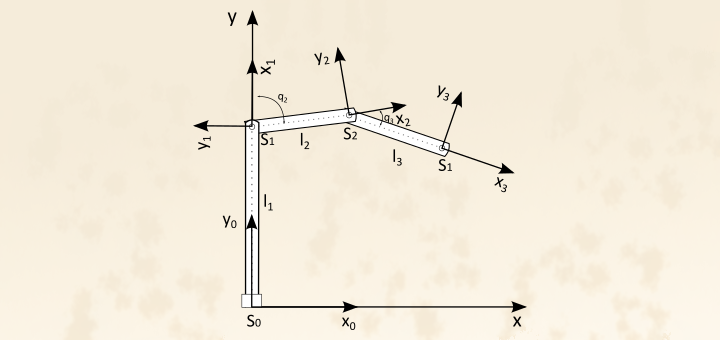


Figura 1 movimientos en un robot

Independientemente del tipo de articulación que utilicemos, lo habitual es nombrar a cada movimiento en una única dimensión mediante el término “Grado de Libertad” (GDL). Con esto podemos definir nuestro robot con base al número de grados de libertad y que para la figura 1 del ejemplo tendríamos un robot con 3 grados de libertad. En el caso de una articulación de rótula tendríamos también 3 grados de libertad (Una rotación en el eje X, otra en el eje Y, y una última en el eje Z).

Para aclarar un poco más el asunto, existen varias formas de definir un robot, siendo el número de grados de libertad uno de ellos, pero también podríamos por su configuración o por su generación. Por otro lado, tenemos que tener claro que cada articulación será movida por un actuador, que para resumir será un motor.

Para aclarar un poco más el asunto, existen varias formas de definir un robot, siendo el número de grados de libertad uno de ellos, pero también podríamos por su configuración o por su generación. Por otro lado, tenemos que tener claro que cada articulación será movida por un actuador, que para resumir será un motor.

## Cinemática del robot

La cinemática del robot se centra en el estudio del movimiento sin tener en cuenta las fuerzas que intervienen en dicho movimiento.

Bajo esta definición se dividen dos tipos de problemas, la Cinemática Directa (DK) y la Cinemática Inversa (IK).

 El primero de los problemas consiste en calcular que posición, en las dimensiones en las que nos movamos, se encontrará el extremo de nuestro robot para cada movimiento de nuestras articulaciones. Con los correspondientes cálculos, podremos conocer en todo momento donde se encuentra cada una de nuestros eslabones.

Como se podrán imaginar, y poniendo como ejemplo un simple brazo robótico que coge tiene que coger una pieza, es fundamental conocer en qué posición nos encontramos. Si no fuese así, no sabríamos si nuestra pinza deberá sujetar la pieza.

 El otro problema es el de la **cinemática inversa**, y que consiste en averiguar cómo colocar nuestros eslabones y articulaciones para que el extremo llegue a una posición concreta. Si lo pensamos un poco, este es un problema muy complejo de resolver ya que, dependiendo del número de grados de libertad, el número de posiciones llegará a ser elevado y por tanto muy difícil de calcular.

Para brazos con un par de grados de libertad, la resolución se puede realizar mediante trigonometría y el teorema de los triángulos semejantes, pero para un mayor número de articulaciones deberemos utilizar otras técnicas más complejas como las matrices de transformación homogéneas.

### ¿Necesito saber matemáticas para mover mi brazo robótico?

Pues, sería muy recomendable si queremos dotarlo de algo parecido a inteligencia, sin embargo, es posible realizar los movimientos mediante aprendizaje o repetición.

La programación por guiado o aprendizaje consiste en realizar la tarea o movimiento mediante el uso de una maqueta del robot o un mando para ser registrado cada posición. Una vez generado el movimiento, este se repite para hacer el trabajo deseado.

## Dinámica del robot

La dinámica se centra en la fuerza necesaria para realizar el movimiento. Para ello se han de tener en cuenta detalles tan importantes como la fricción, la potencia de los motores, la gravedad, la masa del propio brazo robótico o el objeto que deseemos mover.

 Para realizar el movimiento con la suficiente suavidad, se suelen utilizar controladores PID. Estos controladores disponen de 3 tipos de parámetros:

* Proporcional (P)
* Integral (I)
* Derivado (D)

**Simulación de Robots con Gazebo**

Aunque el simulador parece compatible con Windows, las pruebas de compilación sobre un Windows 10 no han sido exitosas y por tanto he optado por aplicar la simulación en un Linux, concretamente en un Ubuntu 16. Aun así, si estas interesados (tercos y aferrados) en trabajar sobre Windows, les recomiendo que instalen una máquina virtual con un Linux, ya que es gratuito y fácil de preparar. (Aunque hay multitud de tutoriales en internet, hay que revisarlos y ver los problemas a los que se afrontan, y como resolverlos).

A la hora de trabajar con nuestro robot, necesitaremos varias aplicaciones y utilidades. En el siguiente vídeo pueden ver como instalar, desde 0, todos las herramientas.

https://youtu.be/an6hjYz2hJY

**Gazebo**

Trabajaremos sobre la última versión de Gazebo, para ellos simplemente deberemos seguir las instrucciones de la dirección

<http://gazebosim.org/tutorials?tut=install_ubuntu&cat=install>

**ROS**

Instalar la última versión de ROS (de preferencia), en este caso ROS siguiendo las instrucciones de <http://wiki.ros.org/Installation/Ubuntu>

**Catkin**

Para realizar, programar y compilar nuestros controladores necesitaremos CATKIN. La instalación se encuentra en <http://wiki.ros.org/catkin>

Para empezar a trabajar, necesitaremos crear un espacio de trabajo (Workspace) para nuestro proyecto, para lo cual seguiremos los siguientes pasos:

Lanzamos la configuración de ROS que en nuestro caso la tendremos en la carpeta /opt/ros/kinetic/setup.bash

$ source /opt/ros/indigo/setup.bash

Creamos la carpeta del espacio de trabajo y la inicializamos

$ mkdir -p ~/catkin\_ws

$ mkdir -p ~/catkin\_ws/src

$ cd ~/catkin\_ws/src

$ catkin\_init\_workspace

Compilamos los ejemplos

$ cd ~/catkin\_ws/

$ catkin\_make

Cuando queramos lanzar nuestro trabajo deberemos tener configurado los diferentes parámetros del sistema, esto lo haremos mediante el comando

$ source ~/catkin\_ws/devel/setup.bash

**Emacs / Vim / neoVim / Geany**

Necesitaremos un editor de texto para poder modificar nuestros ficheros y generar el código. En mi caso utilizaré el Emacs, pero serviría cualquier editor de texto.

# Simulación de Robots con Gazebo. Construcción del robot

Veremos cómo montar nuestro robot.

<https://youtu.be/uI406H8YxCA>

Antes de comenzar con el desarrollo del robot, deberemos configurar nuestro entorno de trabajo. Para ello deberemos crear una carpeta en la que alojaremos nuestro código y en la que estarán localizadas las librerías de compilación.

El primer paso será crear las carpetas mediante el comando “mkdir”. En nuestro caso escogeremos la carpeta de raíz de usuario “**~/**”. La tilde de la ñ se obtiene con la combinación de teclas alt+124 o pulsando alt gr + 4. Otra opción es poner **/home/”usuario”** (lo que va en comillas, es el nombre que cada quien le puso de usuario a su sistema Linux)

Crearemos una carpeta inicial (yo la llamare “Robot”) y dentro de esta crearemos la carpeta “src” (de source que significa fuente en inglés).

Ahora deberemos configurar el terminal para que conozca la ubicación de ROS, para ello ejecutaremos el comando “source /opt/ros/kinetic/setup.bash”.

Ahora inicializamos catkin y compilamos.

cd ~/Robot/src/

catkin\_init\_workspace

cd ..

catkin\_make

Ahora ya estamos en disposición de comenzar a desarrollar nuestro robot. Para ello empezaremos creando dentro de src nuestro proyecto. Esto lo hacemos utilizando el comando “catkin\_create\_pkg” en el que podemos indicar las librerías que vallamos a necesitar. En nuestro caso necesitaremos:

* gazebo\_plugins
* gazebo\_ros
* gazebo\_ros\_control
* ros\_controllers
* roscpp
* std\_msgs

El siguiente paso es construir nuestro mundo. Mi recomendación es que lances Gazebo y directamente guardes un mundo en blanco. Para ello crea una carpeta en el que guardar el mundo y dale un nombre reconocible. En mi caso lo he llamado “miRobot.world” y lo he guardado en la carpeta “worlds”.

Para lanzar nuestro proyecto, utilizaremos el comando “roslaunch”. Este comando necesita un fichero de configuración en el que indicaremos como queremos lanzar nuestra simulación. El fichero lo crearemos en una carpeta dentro la carpeta “worlds” y con el nombre “miRobot.launch”

<launch>

                <arg name="paused" default="false"/>

                <arg name="use\_sim\_time" default="true"/>

                <arg name="gui" default="true"/>

                <arg name="headless" default="false"/>

                <arg name="debug" default="false"/>

                <include file="$(find gazebo\_ros)/launch/empty\_world.launch">

                               <arg name="world\_name" value="$(find miRobot)/worlds/miRobot.world"/>

                               <arg name="verbose" value="true"/>

                               <arg name="debug" value="$(arg debug)" />

                               <arg name="gui" value="$(arg gui)" />

                               <arg name="paused" value="$(arg paused)"/>

                               <arg name="use\_sim\_time" value="$(arg use\_sim\_time)"/>

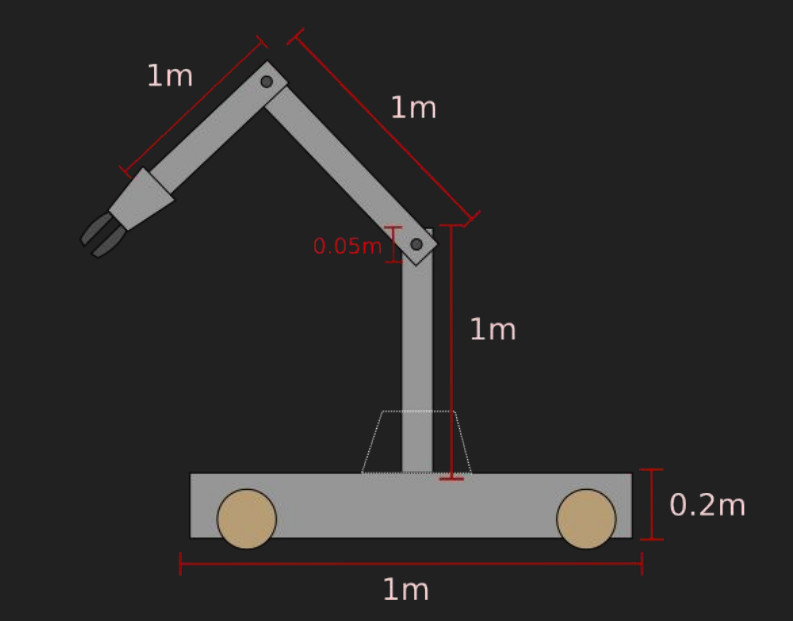
                               <arg name="headless" value="$(arg headless)"/>

                </include>

</launch>

Con las opciones anteriores arrancaremos nuestro robot con el mundo pausado de tal forma que no se aplicará movimiento y físicas a nuestro robot.

**Definición del robot**



Para definir el robot utilizaremos el lenguaje SDF (<http://www.sdformat.org>). En este lenguaje se definirá el robot mediante el elemento Model.

Dentro de este elemento se utilizarán, básicamente, otros 2 elementos, el enlace y la unión.

### Link

El link permite definir el cómo se deberá mostrar el eslabón, cuáles son los criterios de colisión y cuál es la dinámica aplicada.

En el siguiente trozo del código de definición se puede ver:

* La posición que ocupa el brazo “Pose”.
* La inercia que tiene junto con su masa.
* El área de colisión. Este puede ser un objeto básico que encierre al objeto visual, o en nuestro caso es un modelo un poco más elaborado. Además de la geometría de colisión, se encuentran parámetros de superficie como la fricción, el contacto, etc.
* Otro dato importante define como se deberá visualizar el elemento. Esto se encuentra en el tag Visual.

Todos estos parámetros son procesados por Ogre (http://www.ogre3d.org/about/features) y por ODE (http://www.ode.org/) para simular un entorno real.

<link name='arm\_link\_0'>

  <pose frame=''>0.143 0 0.046 0 -0 0</pose>

  <inertial>

       <pose frame=''>0 0 0 0 -0 0</pose>

       <inertia>

            <ixx>0.01</ixx>

            <ixy>0</ixy>

            <ixz>0</ixz>

            <iyy>0.01</iyy>

            <iyz>0</iyz>

            <izz>0.01</izz>

       </inertia>

       <mass>0.845</mass>

  </inertial>

  <collision name='arm\_link\_0\_geom'>

       <pose frame=''>0 0 0 0 -0 0</pose>

       <geometry>

            <mesh>

                 <uri>model://youbot/meshes/arm/arm0\_convex.stl</uri>

            </mesh>

       </geometry>

       <surface>

            <friction>

                 <ode>

                     <mu>0</mu>

                     <mu2>0</mu2>

                     <fdir1>0 0 0</fdir1>

                     <slip1>0</slip1>

                     <slip2>0</slip2>

                 </ode>

                 <torsional>

                     <ode/>

                 </torsional>

            </friction>

            <bounce>

                 <restitution\_coefficient>0</restitution\_coefficient>

                 <threshold>0</threshold>

            </bounce>

            <contact>

                 <ode>

                     <soft\_cfm>0</soft\_cfm>

                     <soft\_erp>0.2</soft\_erp>

                     <kp>1e+13</kp>

                     <kd>1e+11</kd>

                     <max\_vel>-1</max\_vel>

                     <min\_depth>0</min\_depth>

                 </ode>

            </contact>

       </surface>

       <laser\_retro>0</laser\_retro>

       <max\_contacts>10</max\_contacts>

  </collision>

  <visual name='arm\_link\_0\_geom\_visual'>

       <pose frame=''>0 0 0 0 -0 0</pose>

       <geometry>

            <mesh>

                 <uri>model://youbot/meshes/arm/arm0.dae</uri>

            </mesh>

       </geometry>

       <material>

            <script>

                 <uri>model://youbot/materials/scripts/youbot.material</uri>

                 <name>youbot/DarkGrey</name>

            </script>

       </material>

  </visual>

  <gravity>1</gravity>

  <self\_collide>0</self\_collide>

  <kinematic>0</kinematic>

</link>

### Joint

Los Joint (Uniones) son uno de los elementos más importantes para definir nuestro movimiento. Estos definen como están unidos jerárquicamente los eslabones y cuál es el tipo de articulación y actuador definidos en la unión.

* Por un lado, tenemos la jerarquía mediante la definición del padre y el hijo. En el caso de nuestra primera unión tiene como padre “base” con el objetivo de fijar el brazo robótico a la base del robot.
* El siguiente punto son los ejes. SDF permite definir 2 ejes de movimiento, aunque en el robot de este proyecto solo definirá un eje por unión. Los ejes podrán moverse de forma rotacional o prismática y permiten definir:
  + La dinámica del eje.
  + Los límites tanto del ángulo máximo como mínimo que puede adquirir el eje como de la fuerza y velocidad máxima que puede aplicarse.
* Por último, definirá sobre el o los ejes de coordenadas sobre los que puede rotar o desplazarse el eje.

<joint name='arm\_joint\_0' type='revolute'>

  <pose frame=''>0 0 0 0 -0 0</pose>

  <parent>base</parent>

  <child>arm\_link\_0</child>

  <axis>

       <dynamics>

            <damping>0</damping>

            <friction>0</friction>

            <spring\_reference>0</spring\_reference>

            <spring\_stiffness>0</spring\_stiffness>

       </dynamics>

       <limit>

            <lower>0</lower>

            <upper>0</upper>

            <effort>0</effort>

            <velocity>0</velocity>

       </limit>

       <xyz>0 0 1</xyz>

       <use\_parent\_model\_frame>1</use\_parent\_model\_frame>

  </axis>

</joint>

Por último, para lanzar el robot utilizaremos el siguiente comando:

$roslaunch miRobot miRobot.launch

.