



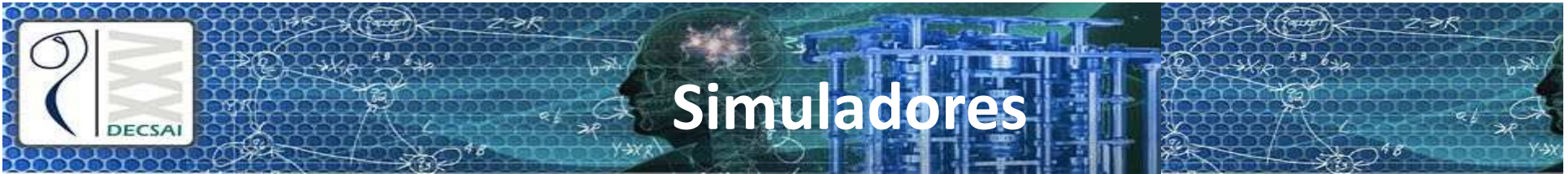
Técnicas de los Sistemas Inteligentes

Curso 2016-17

Práctica1: Robótica.

Sesion2. Simulación en Gazebo

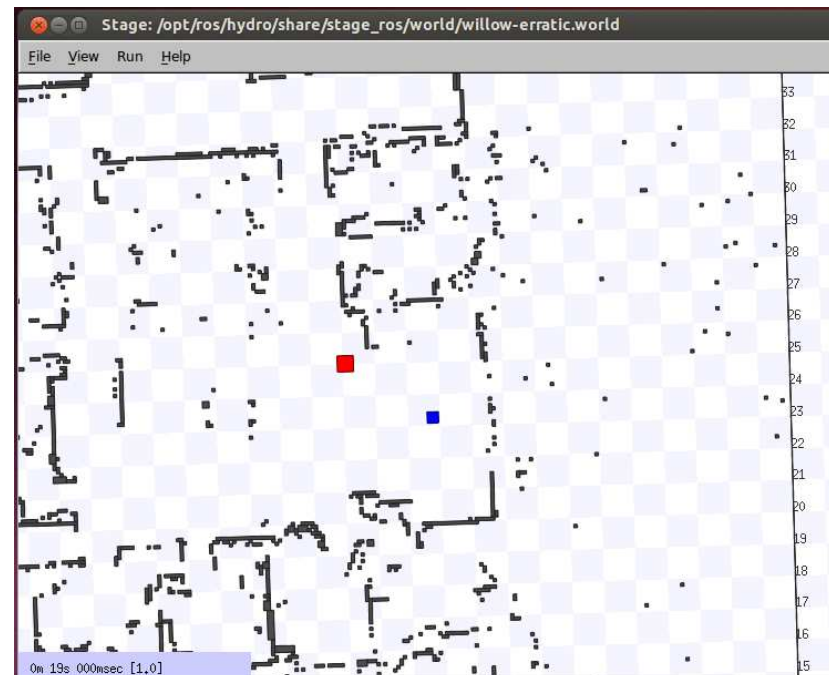
Presentación basada en <http://u.cs.biu.ac.il/~yehoshr1/89-685/>
(C)2013 Roi Yehoshua



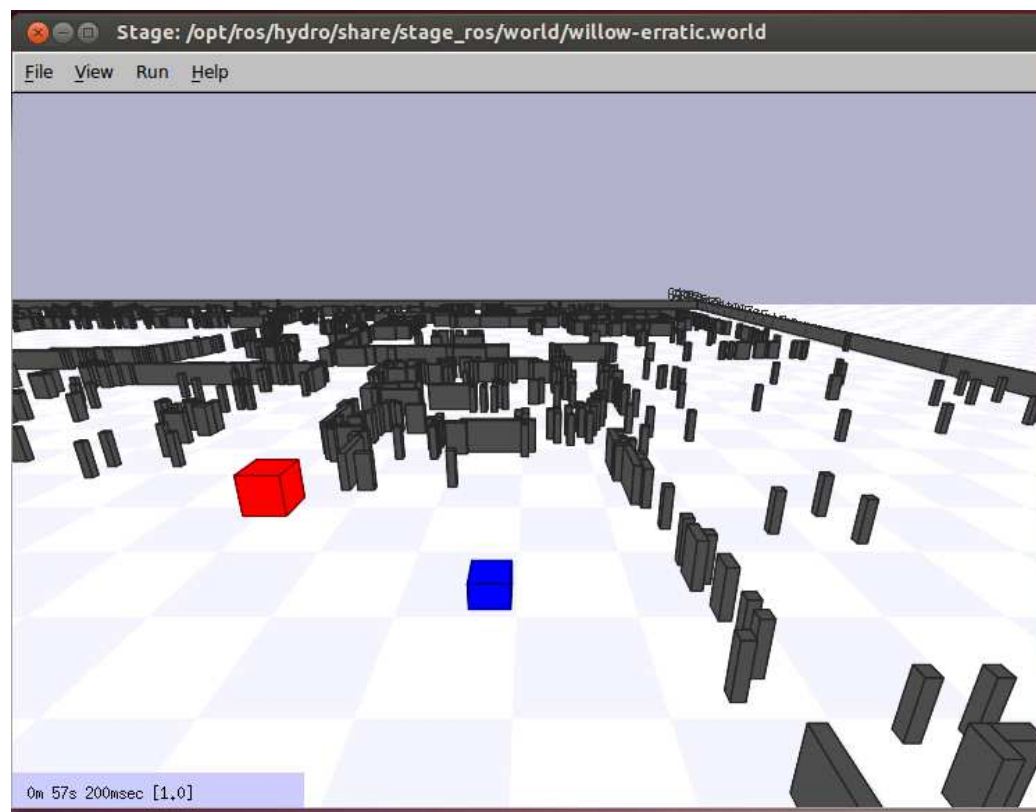
- Con un simulador podemos modelar tanto o tan poco de la realidad como deseemos.
- Los sensores y actuadores pueden modelarse como dispositivos ideales o pueden incorporar varios niveles de distorsión, error o fallos inesperados
- La comprobación automática de algoritmos de control requiere típicamente robots simulados, ya que estos algoritmos necesitan experimentar las consecuencias de sus acciones.
- Debido a su filosofía distribuida y de acoplamiento débil entre módulos proporcionada por la interfaz de mensajes de ROS, una gran mayoría de software de robots puede ejecutarse idénticamente en un robot real o simulado.



- http://wiki.ros.org/stage_ros?distro=hydro
- Un simulador que provee un mundo virtual poblado por objetos, robots móviles y sensores. Los objetos pueden se detectados y manipulados por los robots.



Vista en perspectiva





GAZEBO

- Un simulador multi-robot
- Como Stage, puede simular una población de robots, sensores y objetos, pero en 3D.
- Incluye una simulación precisa de física de cuerpos rígidos y genera feedback de sensores realista.
- Permita que el código diseñado para operar con un robot físico se ejecute en un entorno artificial.
- Gazebo está en desarrollo continuo en la OSRF (Open Source Robotics Foundation)



- [Gazebo Demo](#)



- ROS Indigo viene con Gazebo V2.2
- Gazebo home page - <http://gazebosim.org/>
- Gazebo tutorials - <http://gazebosim.org/tutorials>

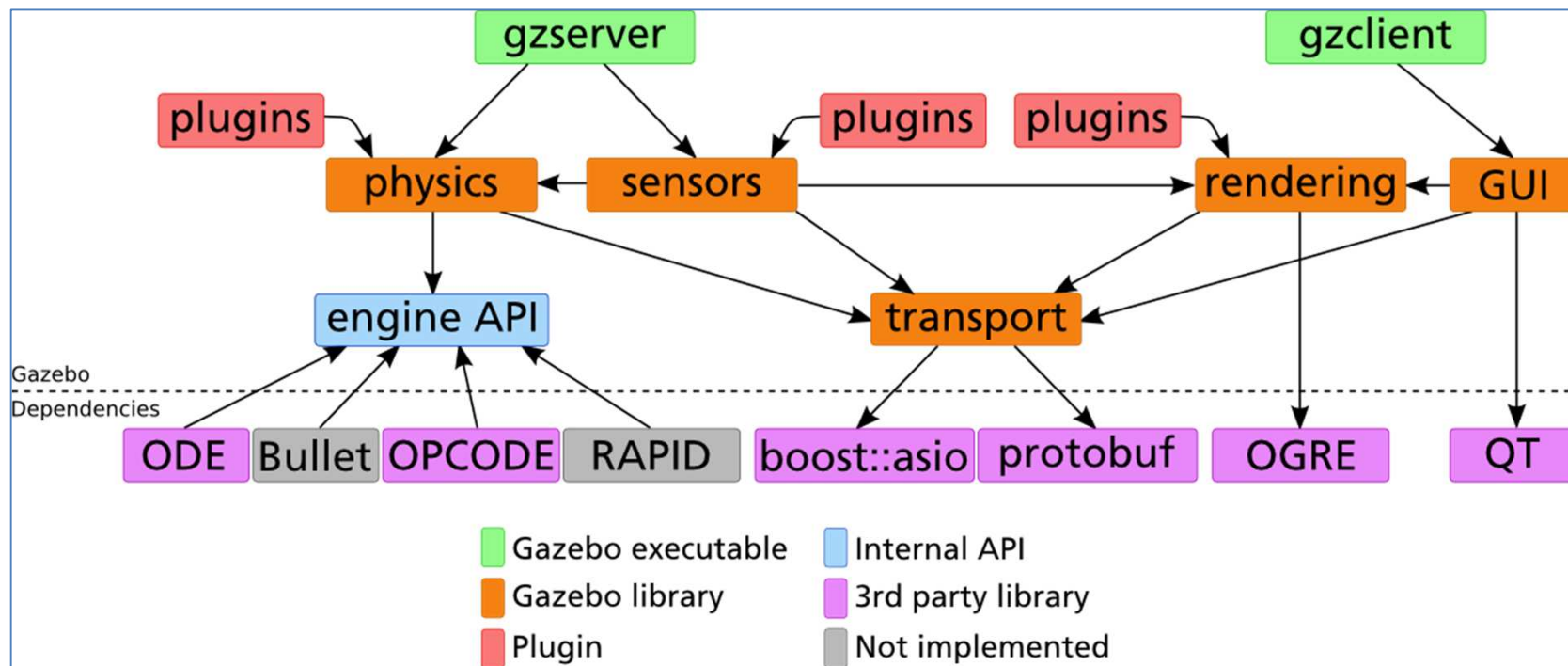


Arquitectura de Gazebo

Gazebo consiste en dos procesos:

- **Server:** Ejecuta el bucle de física y genera datos de sensores
 - *Executable:* gzserver
 - *Libraries:* Physics, Sensors, Rendering, Transport
- **Client:** Proporciona interacción de usuario y visualización de la simulación.
 - *Executable:* gzclient
 - *Libraries:* Transport, Rendering, GUI

Arquitectura de Gazebo





Ejecutar Gazebo desde ROS

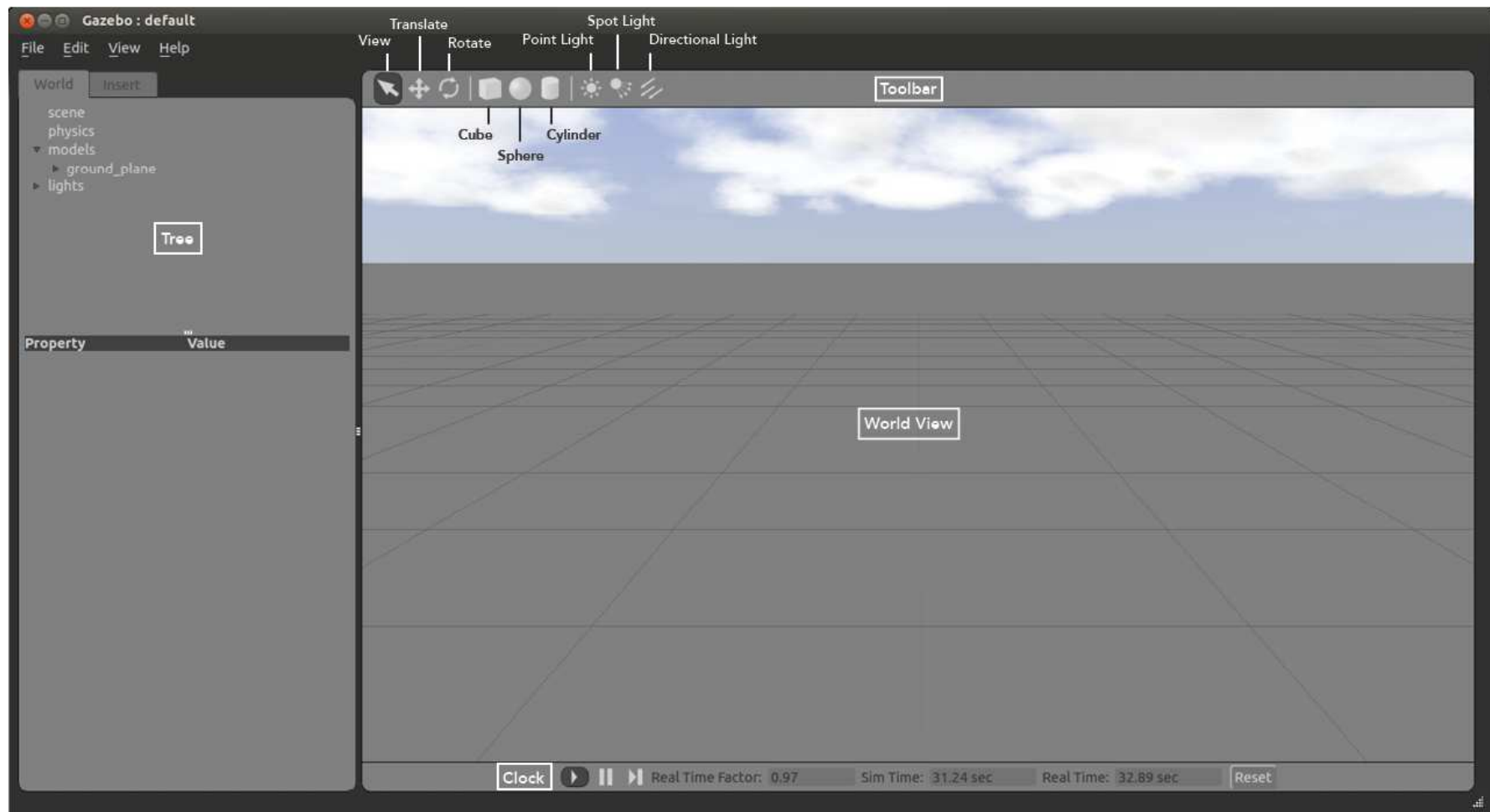
- Para lanzar Gazebo :

```
$ rosrun gazebo_ros gazebo
```

- Nota: Puede tomar minutos para actualizar su base de datos de modelos



Gazebo User Interface





Pestaña “World View”

- La pestaña World View muestra el mundo y todos los modelos visualizados.
- Aquí se pueden añadir, manipular o eliminar modelos
- Podemos intercambiar los modos View, Translate and Rotate (parte izda. de la barra de herramientas)

View Mode

Translate	Left-press + drag
Orbit	Middle-press + drag
Zoom	Scroll wheel
Accelerated Zoom	Alt + Scroll wheel
Jump to object	Double-click object
Select object	Left-click object

Translate Mode

Translate	Left-press + drag
Translate (x-axis)	Left-press + X + drag
Translate (y-axis)	Left-press + Y + drag
Translate (z-axis)	Left-press + Z + drag

(Orbit & Zoom work in this mode, as well)

Rotate Mode

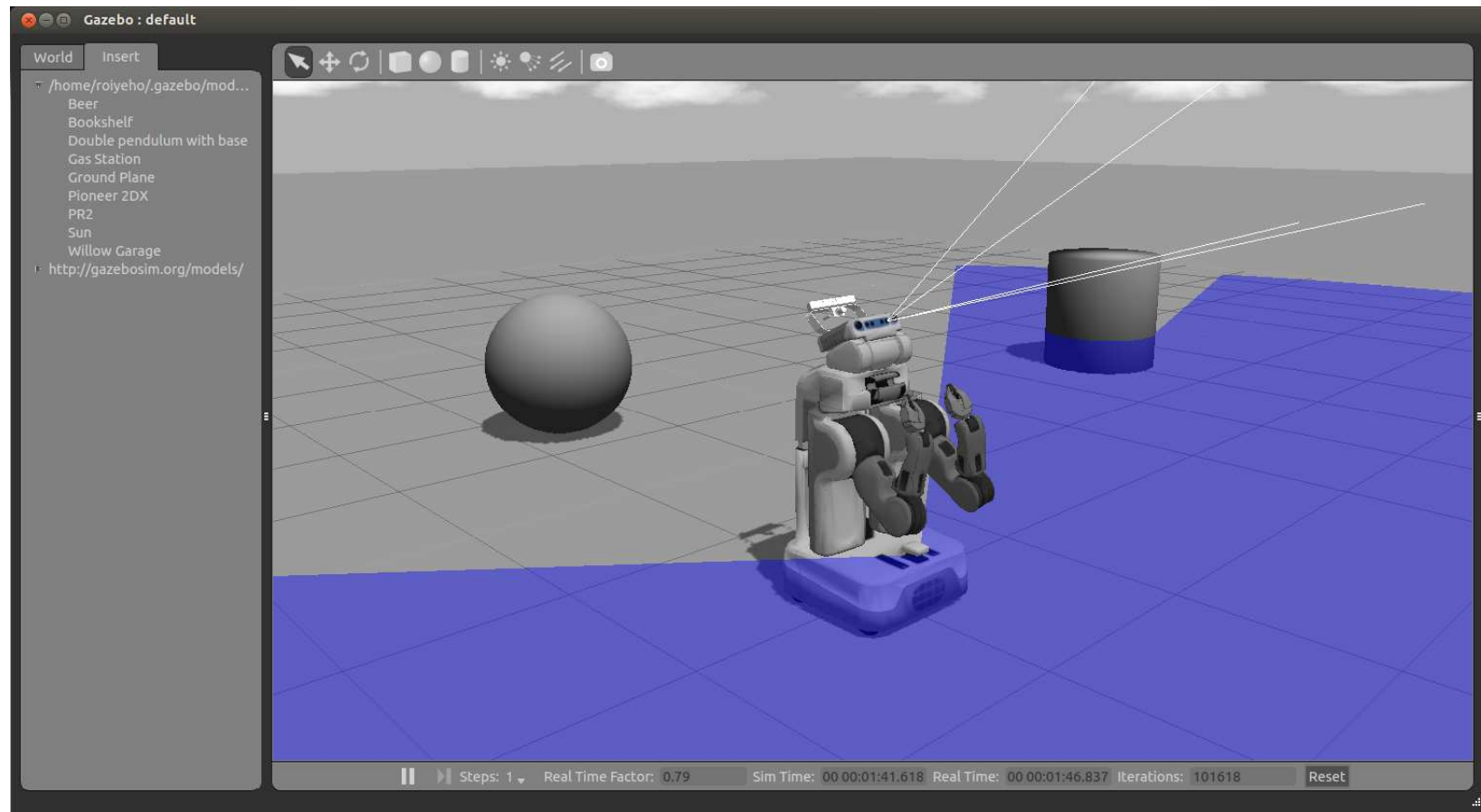
Rotate (spin) object	Left-press + drag
Rotate (x-axis)	Left-press + X + drag
Rotate (y-axis)	Left-press + Y + drag
Rotate (z-axis)	Left-press + Z + drag

(Orbit & Zoom work in this mode, as well)



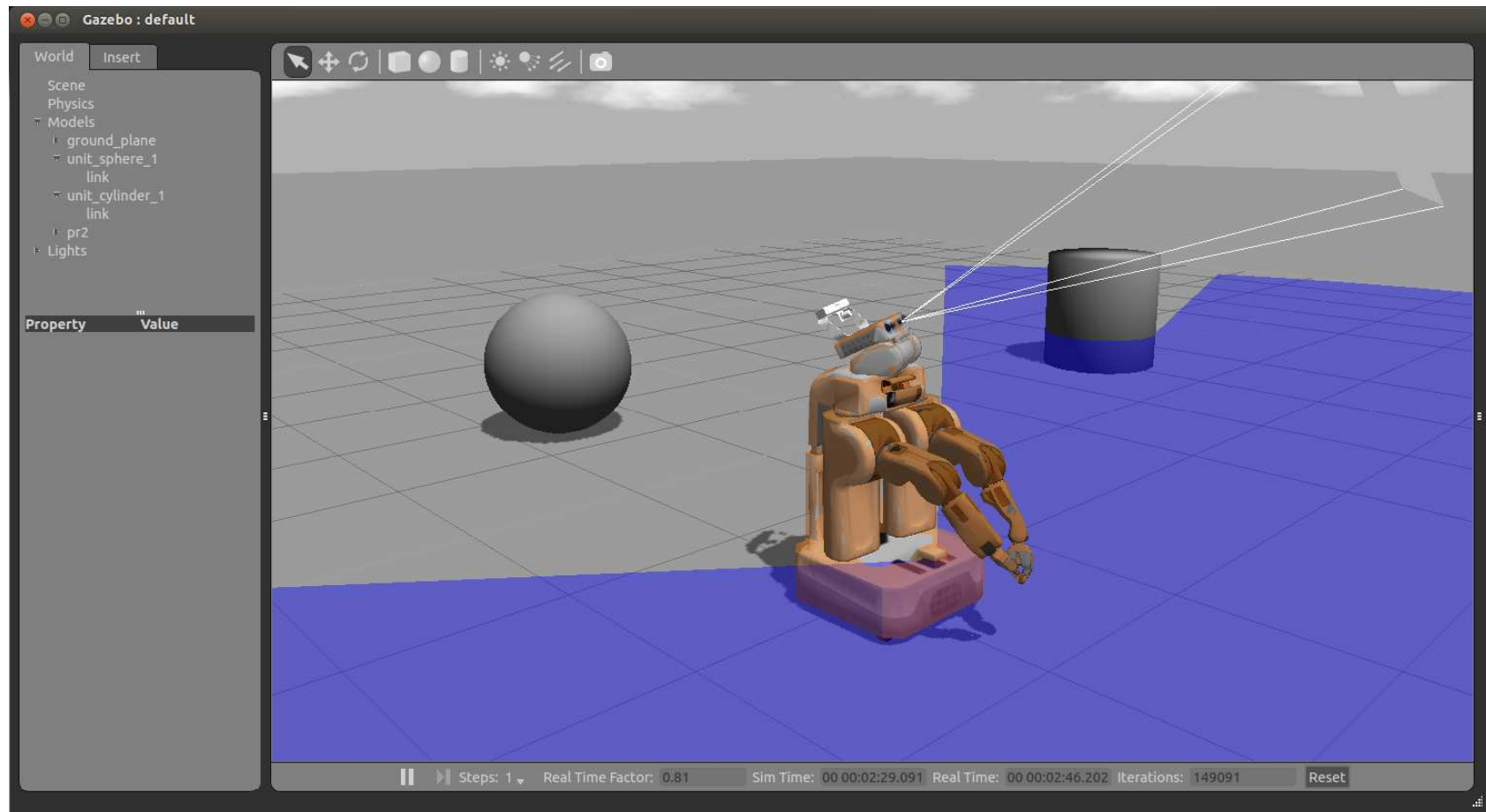
To add a model to the world:

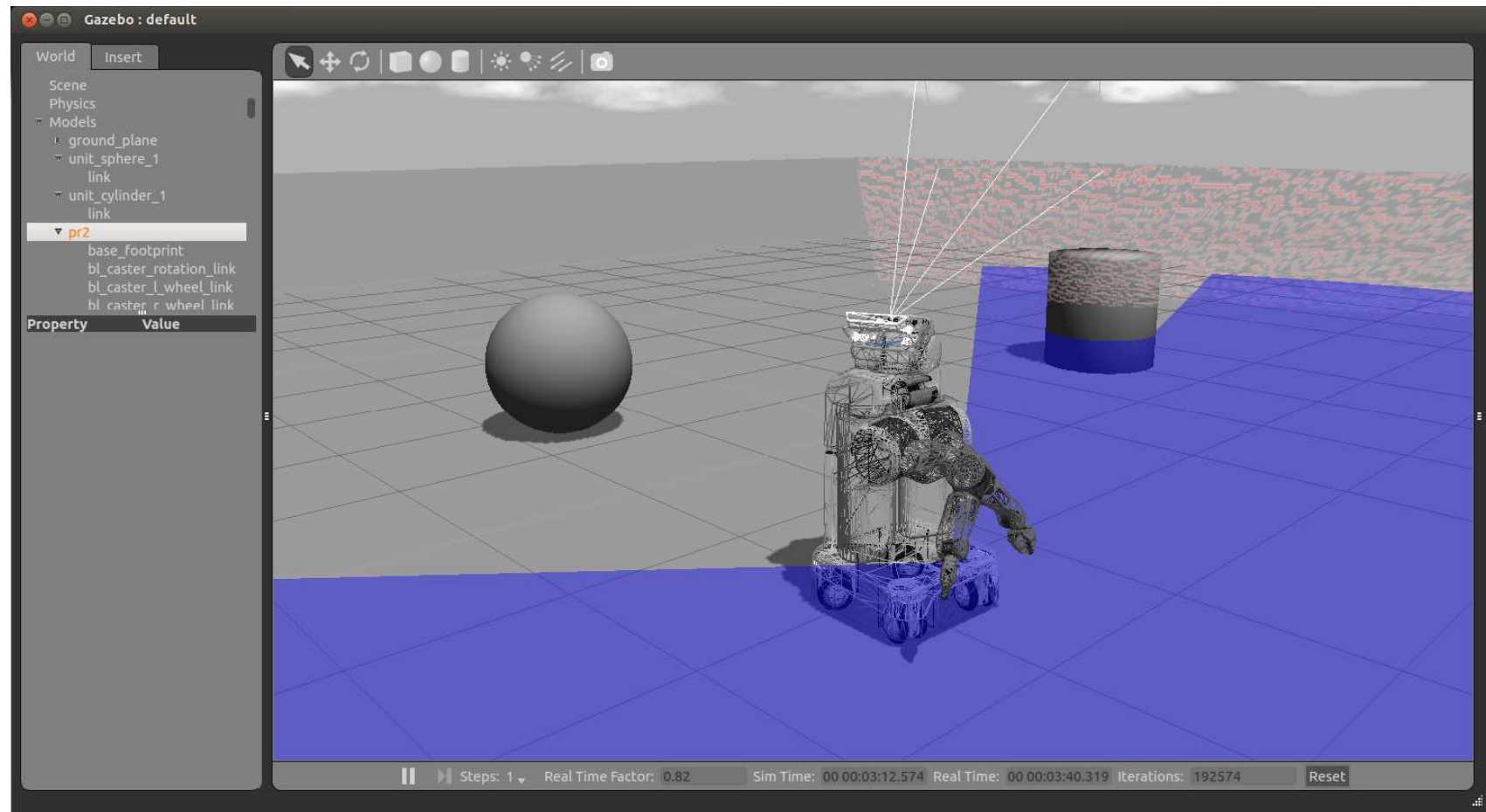
- left-click on the desired model in the Insert Tab on the left side
- move the cursor to the desired location in World View
- left-click again to release
- Use the Translate and Rotate modes to orient the model more precisely





- The models item in the world tab contains a list of all models and their links
- Right-clicking on a model in the Models section gives you three options:
 - **Move to** – moves the view to be directly in front of that model
 - **Follow**
 - **View** – allows you to view different aspects of the model, such as Wireframe, Collisions, Joints
 - **Delete** – deletes the model







- El reloj de simulación puede controlarse: pause and step
- Está en la parte inferior de la World View

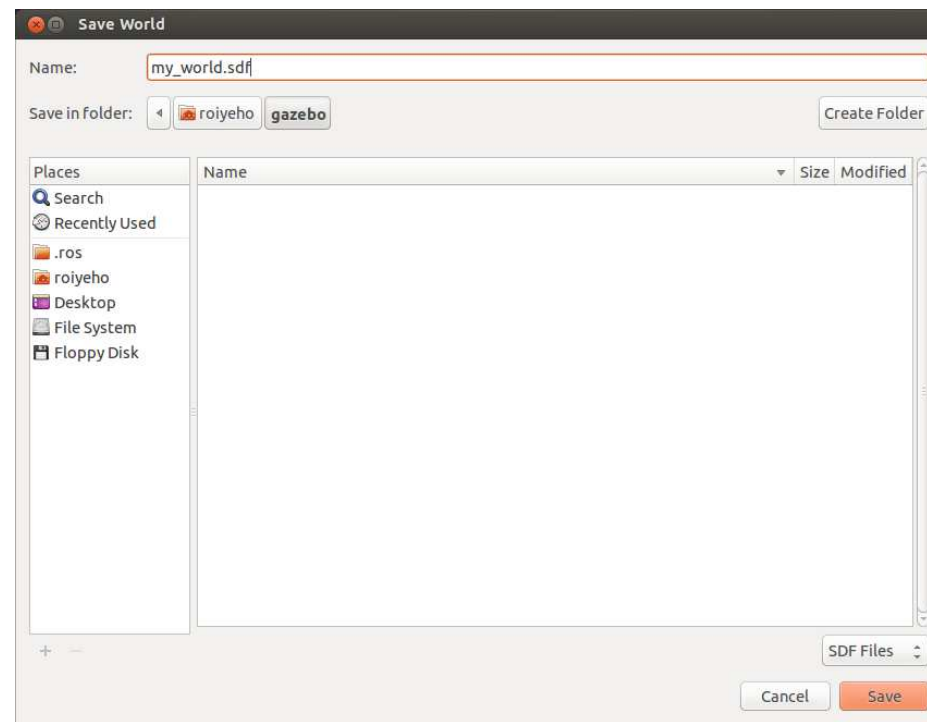


- **Real Time Factor:** muestra cómo de rápido o lento va la simulación en comparación con el tiempo real.
 - Factor < 1.0 -> simulation is running slower than real time
 - Factor > 1.0 -> simulation is running faster than real time



Guardar un mundo

- El mundo actual puede guardarse con File->Save As menu.
- Enter my_world.sdf as the file name and click OK





- Un mundo guardado puede abrirse desde la línea de comandos

```
$ rosrun gazebo_ros gazebo my_world.sdf
```

- Matar los procesos de gazebo antes de ejecutar la línea de arriba.
- The filename must be in the current working directory, or you must specify the complete path



Ficheros de Gazebo

- World description files (.world): describen un mundo, compuesto de varios modelos.
- Launch files (.launch): usados en ROS para lanzar un mundo y configurar valores de simulación.
- Model files: describen un único modelo, pueden incluirse desde ficheros .world.
 - Cada modelo se define a partir de Links, Collisions, Visuals, Inertial, Sensors, Joints, Plugins.



World Description File

- El fichero de descripción del mundo contiene todos los elementos de la simulación: robots, luces, sensores y objetos estáticos.
- Este fichero está en formato “SDF” (un formato XML que describe objetos y entornos para simuladores de robots, visualización y control) <http://sdformat.org/> tiene la extensión .world
- Gazebo server (gzserver) lee este fichero para generar y poblar el mundo.



World Files de ejemplo

- Gazebo viene con varios mundos de ejemplo
- Se encuentran en el directorio `/worlds` en la ruta de los recursos de Gazebo
 - A typical path might be `/usr/share/gazebo-2.2`
- En el paquete **gazebo_ros** hay ficheros **launch** que cargan algunos de estos ejemplos.
- For example, to launch willowgarage_world type:

```
$ roslaunch gazebo_ros willowgarage_world.launch
```



```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>
    <include>
      <uri>model://willowgarage</uri>
    </include>
  </world>
</sdf>
```

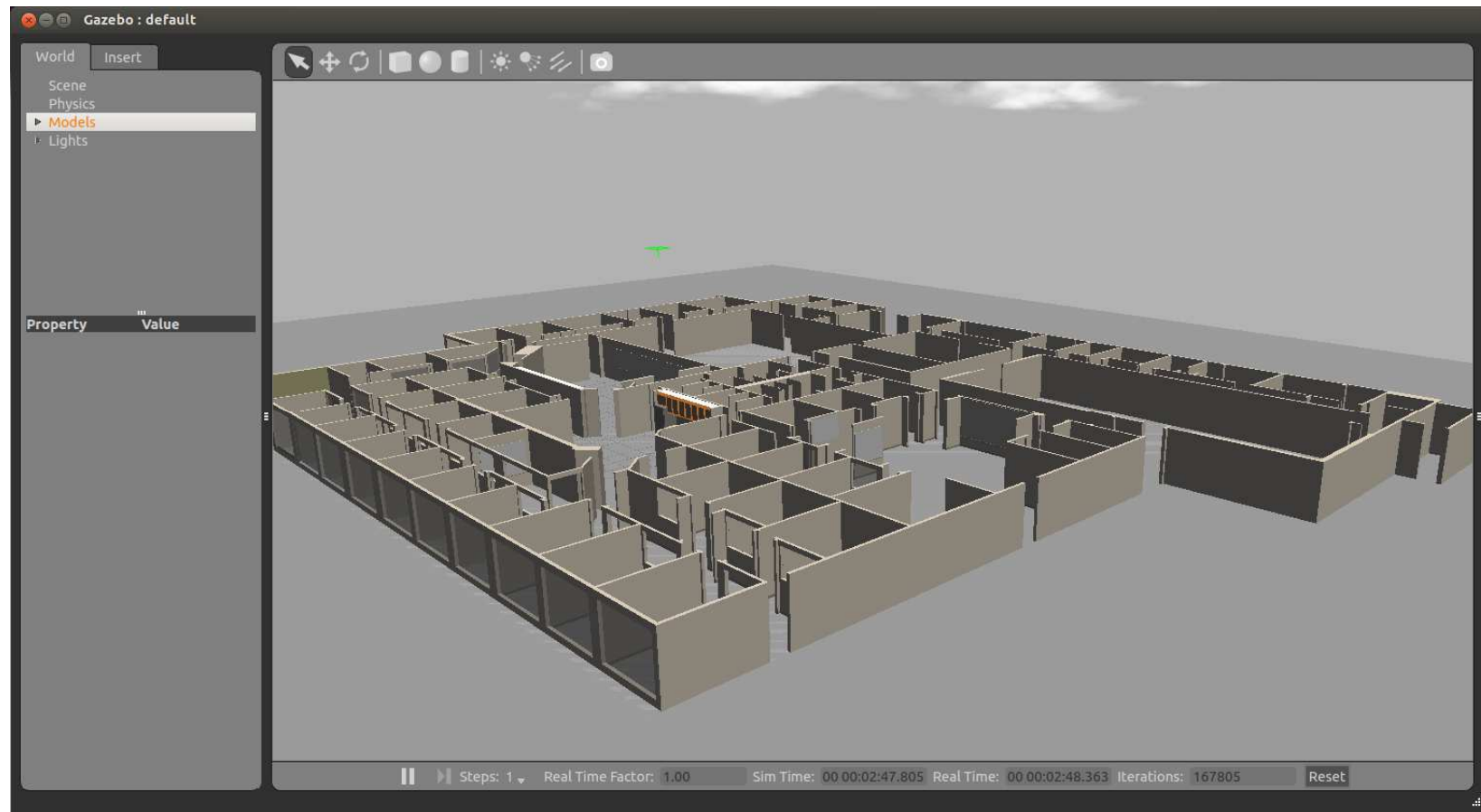
- In this world file snippet you can see that three models are referenced
- The three models are searched for within your local Gazebo Model Database
- If not found there, they are automatically pulled from Gazebo's online database



willowgarage_world.launch

```
<launch>
  <!-- We resume the logic in empty_world.launch, changing only the name of the world to
  be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="worlds/willowgarage.world"/> <!-- Note: the
world_name is with respect to GAZEBO_RESOURCE_PATH environmental variable -->
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>
</launch>
```

- This launch file inherits most of the necessary functionality from empty_world.launch
- The only parameter we need to change is the world_name parameter, substituting the empty.world world file with willowgarage.world
- The other arguments are simply set to their default values





- A model file uses the same SDF format as world files, but contains only a single `<model>` tag
- Once a model file is created, it can be included in a world file using the following SDF syntax:

```
<include filename="model_file_name"/>
```

- You can also include any model from the online database and the necessary content will be downloaded at runtime



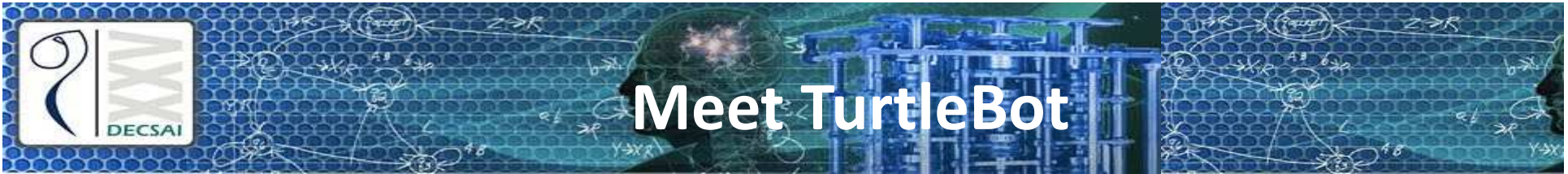
willowgarage Model SDF File


```
<?xml version="1.0" ?>
<sdf version="1.4">
  <model name="willowgarage">
    <static>true</static>
    <pose>-20 -20 0 0 0 0</pose>
    <link name="walls">
      <collision name="collision">
        <geometry>
          <mesh>
            <uri>model://willowgarage/meshes/willowgarage_collision.dae</uri>
          </mesh>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://willowgarage/meshes/willowgarage_visual.dae</uri>
          </mesh>
        </geometry>
        <cast_shadows>>false</cast_shadows>
      </visual>
    </link>
  </model>
</sdf>
```




Components of Models

- **Links:** A link contains the physical properties of one body of the model. This can be a wheel, or a link in a joint chain.
 - Each link may contain many collision, visual and sensor elements
- **Collision:** A collision element encapsulates a geometry that is used to collision checking.
 - This can be a simple shape (which is preferred), or a triangle mesh (which consumes greater resources).
- **Visual:** A visual element is used to visualize parts of a link.
- **Inertial:** The inertial element describes the dynamic properties of the link, such as mass and rotational inertia matrix.
- **Sensor:** A sensor collects data from the world for use in plugins.
- **Joints:** A joint connects two links.
 - A parent and child relationship is established along with other parameters such as axis of rotation, and joint limits.
- **Plugins:** A shared library created by a 3D party to control a model.



- <http://wiki.ros.org/Robots/TurtleBot>
 - Una plataforma de robótica móvil minimalista para para educación y prototipado basados en ROS
 - Tiene una base móvil de guiado diferencial.
 - Sobre la base tiene una pila de bandejas que proporcionan espacio para un portátil, kinect u otros dispositivos
 - No tiene un LIDAR
 - Pero el mapeo y la navegación pueden funcionar bastante bien en espacios de interior.
- 

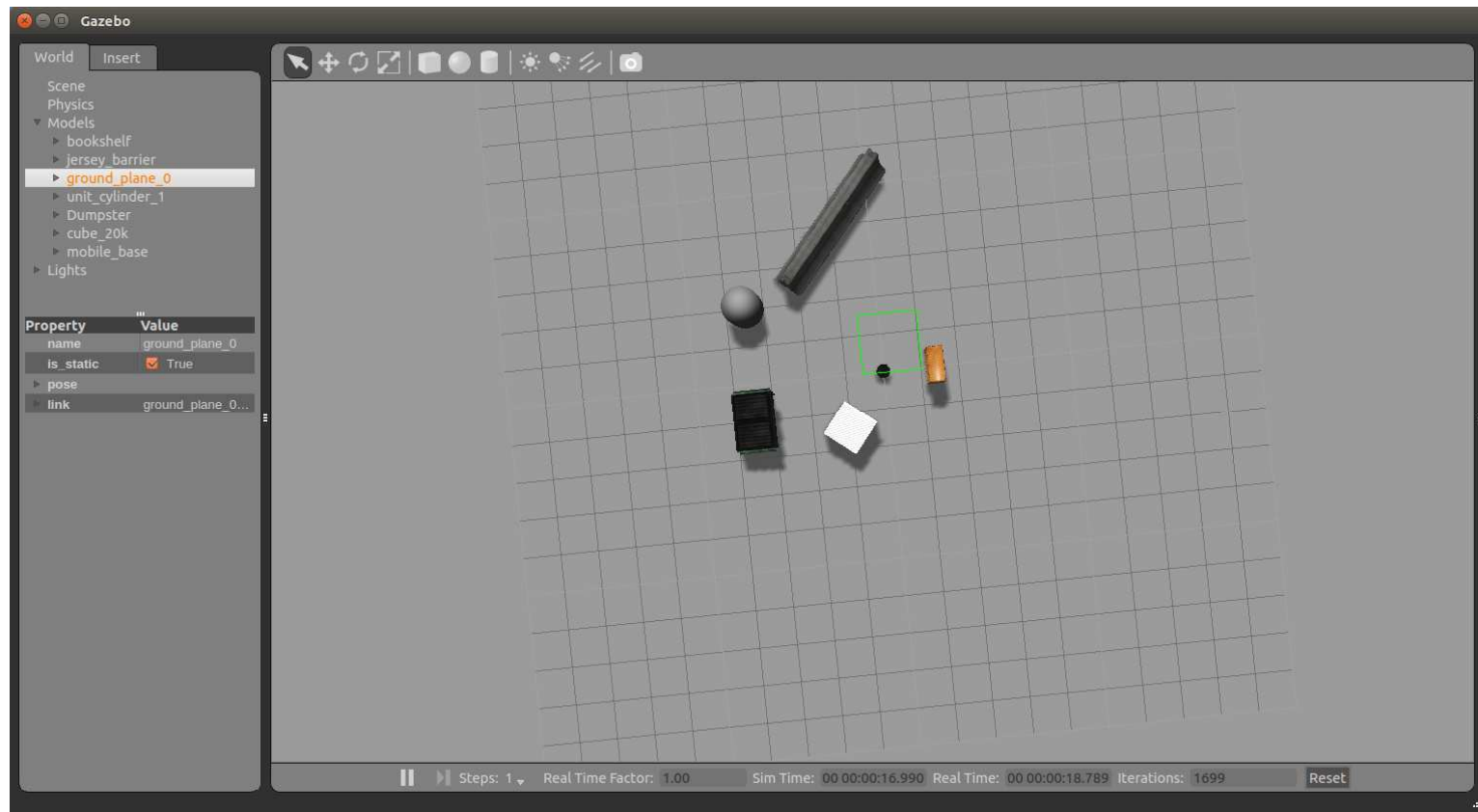


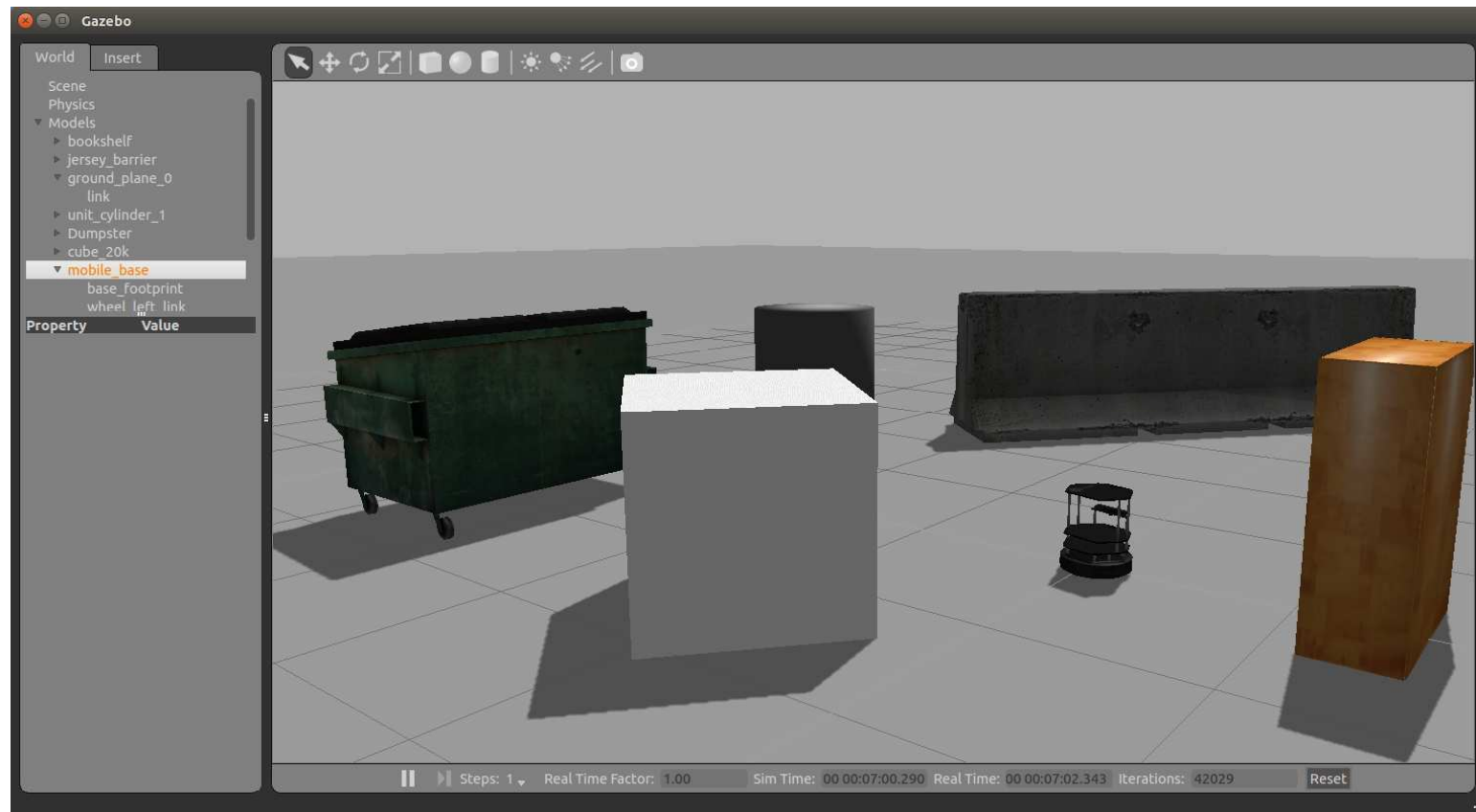
- Instalación del stack de simulación Turtlebot:

```
$ sudo apt-get install ros-indigo-turtlebot-gazebo ros-indigo-turtlebot-apps ros-indigo-turtlebot-rviz-launchers
```

- Lanzar un mundo simulado de Turtlebot:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```







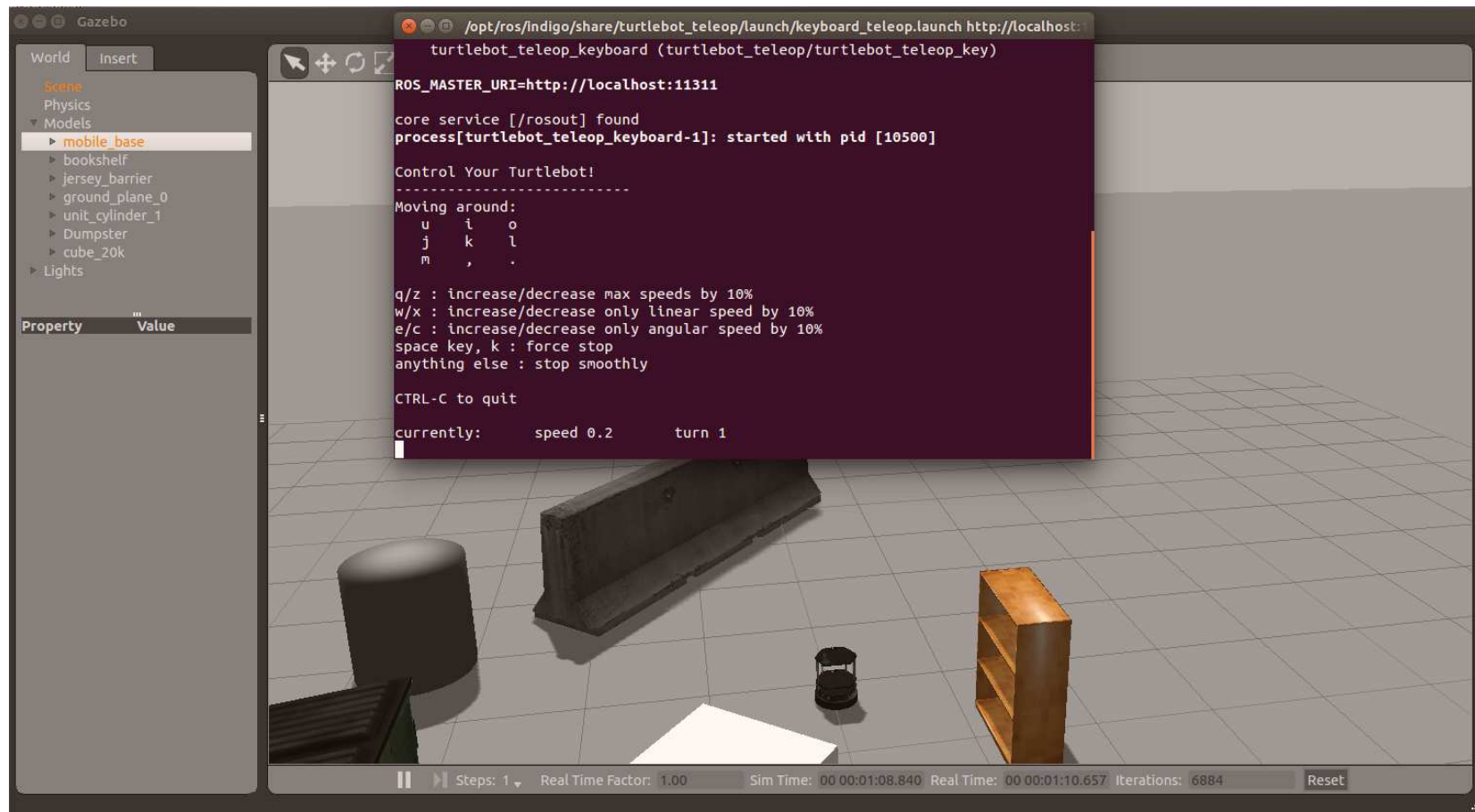
Mover Turtlebot con Teleoperación

- Usando el paquete de teleoperación de Turtlebot para moverlo con el teclado
- Comando:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```



Mover Turtlebot con Teleoperación





¿Cómo lo movemos con un nodo ROS?

- Para controlar un robot en ROS automáticamente, necesitamos conocer algo del ecosistema de nodos, topics y messages que se están usando actualmente.
- Ejecutar `rqt_graph` para comprobar qué nodos y topics están actualmente activos.



Consultando el ecosistema

- ¿Qué topics se están publicando actualmente?

```
$ rostopic list
```

```
juan@UltraJuanitook:~$ rostopic list
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
/clock
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/switch
/cmd_vel_mux/input/teleop
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
/depthImage_to_laserscan/parameter_descriptions
/depthImage_to_laserscan/parameter_updates
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/joint_states
/laserscan_nodelet_manager/bond
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/velocity
/mobile_base/events/bumper
/mobile_base/events/cliff
/mobile_base/sensors/bumper_pointcloud
/mobile_base/sensors/core
/mobile_base/sensors/imu_data
/mobile_base_nodelet_manager/bond
/odom
/rosout
/rosout_agg
/scan
/tf
/tf_static
juan@UltraJuanitook:~$
```

Muestra una lista de topics poco informativa.

En general, para controlar un robot estamos interesados en topics que nos informen sobre **escaneos láser**, que nos permitan enviar **órdenes de movimiento** al robot (comandos de velocidad) o que nos informen sobre la posición (**odometría** del robot).



Consultando el ecosistema

- ¿Qué topics publican información de escaneos láser?
- Primero tenemos que saber qué mensajes se utilizan en ROS para representar un scan laser.
- Podemos consultar la documentación de ros sobre mensajes estándar http://wiki.ros.org/std_msgs y comunes http://wiki.ros.org/common_msgs o usar la herramienta de línea de comandos **rosmmsg**



Consultando el ecosistema

```
$ rosmmsg show LaserScan
```

```
juan@UltraJuanitook: ~  
juan@UltraJuanitook:~$ rosmmsg show LaserScan  
[sensor_msgs/LaserScan]:  
std_msgs/Header header  
  uint32 seq  
  time stamp  
  string frame_id  
float32 angle_min  
float32 angle_max  
float32 angle_increment  
float32 time_increment  
float32 scan_time  
float32 range_min  
float32 range_max  
float32[] ranges  
float32[] intensities
```

El tipo de mensaje de escaneos láser es **sensor_msgs/LaserScan**



Consultando el ecosistema

- Ahora ya podemos saber qué topics publican escaneos láser con la orden

```
$ rostopic find sensor_msgs/LaserScan
```

- (usar el tabulador para encontrar las opciones)

```
juan@UltraJuanitook: ~  
juan@UltraJuanitook:~$ rostopic find sensor_msgs/LaserScan  
/scan  
juan@UltraJuanitook:~$
```

- De esta forma hemos averiguado que la información sobre escaneos láser se publica en el topic **/scan**.
- Podemos proceder de manera análoga para determinar qué topics se usan para publicar información de odometría (rostopic show Odometry, etc...) o de comandos de movimiento (rostopic show Twist). El tipo de mensaje geometry_msgs/Twist se utiliza para enviar velocidades a los motores de un robot.



Datos de Scan Láser

- El tipo de mensaje de un Scan Láser es `sensor_msgs/LaserScan`
- Puede consultarse su estructura con

```
$ rosmg show sensor_msgs/LaserScan
```

- La descripción de cada uno de sus campos la podemos consultar en la url de la siguiente slide.



- http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

```
# Single scan from a planar laser range-finder

Header header
# stamp: The acquisition time of the first ray in the scan.
# frame_id: The laser is assumed to spin around the positive Z axis
# (counterclockwise, if Z is up) with the zero angle forward along the x axis

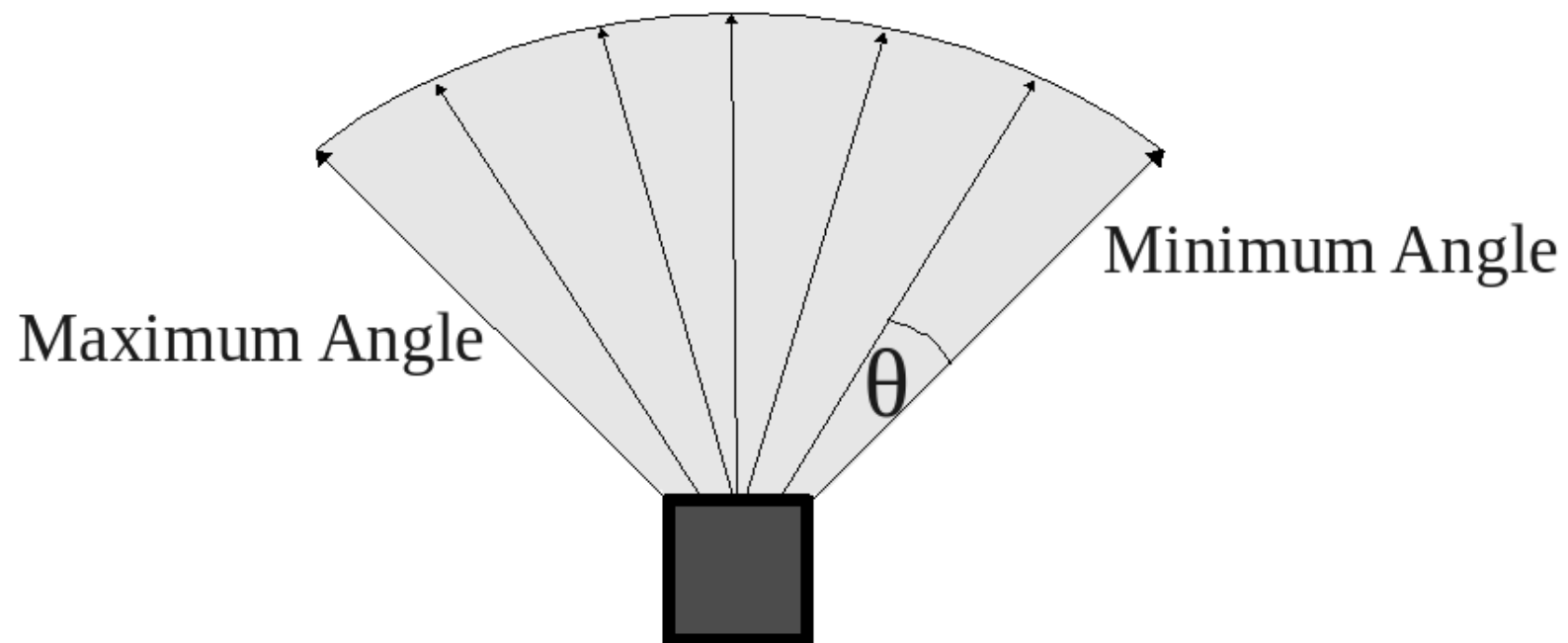
float32 angle_min # start angle of the scan [rad]
float32 angle_max # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment # time between measurements [seconds] - if your scanner
# is moving, this will be used in interpolating position of 3d points
float32 scan_time # time between scans [seconds]

float32 range_min # minimum range value [m]
float32 range_max # maximum range value [m]

float32[] ranges # range data [m] (Note: values < range_min or > range_max should be
discarded)
float32[] intensities # intensity data [device-specific units]. If your
# device does not provide intensities, please leave the array empty.
```

Laser Scanner





- Un sensor laser común usado en robótica

http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx.html

Scanning range finder (SOKUIKI sensor)

A4 size print

URG-04LX

FDA approved
SOKUIKI sensor for intelligent robots



Scanning Laser Range Finder, the best optimized sensor for environment recognition.
Suitable for next generation intelligent robots with an autonomous system and privacy security.

Technical Document >> [Download](#) <<



Model No.	URG-04LX
Power source	5VDC \pm 5%*1
Current consumption	500mA or less(800mA when start-up)
Measuring area	60 to 4095mm(white paper with 70mm \square) 240°
Accuracy	60 to 1,000mm : \pm 10mm, 1,000 to 4,095mm : 1% of measurement
Repeatability	60 to 1,000mm : \pm 10mm
Angular resolution	Step angle : approx. 0.36° (360° /1,024 steps)
Light source	Semiconductor laser diode(λ =785nm), Laser safety class 1(IEC60825-1, 21 CFR 1040.10 & 1040.11)
Scanning time	100ms/scan
Noise	25dB or less
Interface	USB, RS-232C(19.2k, 57.6k, 115.2k, 250k, 500k, 750kbps), NPN open-collector(synchronous output of optical scanner : 1 pce)
Communication specifications	Exclusive command(SCIP Ver.1.1 or Ver.2.0)*2
Ambient temperature/humidity	-10 to +50 degrees C, 85% or less(Not condensing, not icing)
Vibration resistance	10 to 55Hz, double amplitude 1.5mm Each 2 hour in X, Y and Z directions
Impact resistance	196m/s ² , Each 10 time in X, Y and Z directions
Weight	Approx. 160g
Accessory	Cable for power*communication/input*output(1.5m) 1 pce, D-sub connector with 9 pins 1 pce*3



- Example of a laser scan message :
(rostopic echo /scan -n1)

```
roiyeho@ubuntu: ~  
---  
header:  
  seq: 1594  
  stamp:  
    secs: 159  
    nsecs: 500000000  
  frame_id: base_laser_link  
angle_min: -2.35837626457  
angle_max: 2.35837626457  
angle_increment: 0.00436736317351  
time_increment: 0.0  
scan_time: 0.0  
range_min: 0.0  
range_max: 30.0  
ranges: [2.427844524383545, 2.42826247215271, 2.4287266731262207, 2.4292376041412354, 2.429795026779175, 2.430398941  
040039, 2.4310495853424072, 2.4317471981048584, 2.4324913024902344, 2.4332826137542725, 2.4341206550598145, 2.435005  
6648254395, 2.4359381198883057, 2.436917543411255, 2.437944173812866, 2.439018487930298, 2.4401402473449707, 2.44130  
94520568848, 2.4425265789031982, 2.443791389465332, 2.4451043605804443, 2.446465253829956, 2.4478745460510254, 2.449  
3319988250732, 2.450838088989258, 2.452392816543579, 2.453996419906616, 2.455648899078369, 2.457350492477417, 2.4591  
01438522339, 2.460901975631714, 2.462752103805542, 2.4646518230438232, 2.466601848602295, 2.468601942062378, 2.47065  
23418426514, 2.4727535247802734, 2.474905490875244, 2.4771084785461426, 2.479362726211548, 2.481668472290039, 2.4840  
259552001953, 2.4864354133605957, 2.4888970851898193, 2.4914112091064453, 2.4939777851104736, 2.4965975284576416, 2.  
4992706775665283, 2.5019969940185547, 2.504777193069458, 2.5076115131378174, 2.510500192642212, 2.5134434700012207,  
2.516441822052002, 2.5194954872131348, 2.5226047039031982, 2.5257697105407715, 2.5289909839630127, 2.53226900100708,  
2.5356037616729736, 2.5389959812164307, 2.542445659637451, 2.5459535121917725, 2.5495197772979736, 2.55314469337463  
4, 2.5568289756774902, 2.560572624206543, 2.56437611579895, 2.568240165710449, 2.572165012359619, 2.576151132583618,  
2.5801987648010254, 2.584308624267578, 2.5884809494018555, 2.5927164554595947, 2.597015380859375, 2.601378202438354  
5, 2.6058056354522705, 2.610297918319702, 2.6148557662963867, 2.6194796562194824, 2.6241698265075684, 2.628927230834  
961, 2.6337523460388184, 2.63478422164917, 2.6436073780059814, 2.6486384868621826, 2.6537396907806396, 3.44798207283  
02, 3.4547808170318604, 3.461672306060791, 3.4686577320098877, 3.4757378101348877, 3.4829134941101074, 3.49018549919  
1284, 3.4975550174713135, 3.5050225257873535, 3.5125889778137207, 3.5202558040618896, 3.5280232429504395, 3.53589296  
3409424, 3.543865442276001, 3.5519418716430664, 3.5601232051849365, 3.568410634994507, 3.5768051147460938, 3.5853075
```



Depth Image to Laser Scan

- TurtleBot no tiene un LIDAR por defecto
- Pero la imagen de su cámara de profundidad (depth Image Camera) puede usarse como un scaneo láser.
- El nodo [depthimage to laserscan](#) toma una imagen de profundidad y genera un scan láser 2D basado en los parámetros proporcionados
- Este nodo se ejecuta automáticamente cuando ejecutamos Turtlebot simulado en Gazebo.
- Los valores del array que representa las muestras láser contienen NaNs y +-Infs (cuando el rango es menor que range_min o mayor que range_max)
 - Comparaciones con NaNs siempre devuelven false



Depth Image to Laser Scan

- turtlebot_world.launch

```
<launch>
...
<!-- Fake laser -->
<node pkg="nodelet" type="nodelet" name="laserscan_nodelet_manager"
args="manager"/>
<node pkg="nodelet" type="nodelet" name="depthimage_to_laserscan"
  args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet
laserscan_nodelet_manager">
  <param name="scan_height" value="10"/>
  <param name="output_frame_id" value="/camera_depth_frame"/>
  <param name="range_min" value="0.45"/>
  <remap from="image" to="/camera/depth/image_raw"/>
  <remap from="scan" to="/scan"/>
</node>
</launch>
```




- Example of a laser scan message from TurtleBot:
(rostopic echo /scan -n1)

[illegible]



Wander-bot

- Vamos a juntar todos los conceptos que hemos aprendido hasta ahora para crear un robot que puede deambular en su entorno
- Esta es una tarea que, no siendo terriblemente espectacular, realizan por ejemplo (con algún matiz) las aspiradoras inteligentes.



- Comenzaremos con un nodo llamado **stopper** que hará que el robot se mueva hacia adelante hasta que detecta que está demasiado cerca de un obstáculo.
- Usaremos el sensor láser (adaptado como hemos visto) para detectar el obstáculo y enviaremos valores de velocidad lineal para que se mueva.
- Crear un paquete llamado **wander_bot**

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg wander_bot std_msgs rospy roscpp
```



- Copiar los ficheros (descargándolos desde PRADO en el fichero stopper.zip)
 - Stopper.h
 - Stopper.cpp
 - run_stopper.cpp (main function)
- En el directorio src del paquete wander_bot



- Vamos a implementar un nodo que guía al robot hasta que choca con un obstáculo.
- Para ello necesitamos publicar mensajes tipo ***Twist messages*** bajo el topic **`cmd_vel/input/teleop`** (al que está suscrito Turtlebot)
 - Este *topic* es el responsable de enviar órdenes de velocidad al robot.



- http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html
- This message has a **linear** component for the (x,y,z) velocities, and an **angular** component for the angular rate about the (x,y,z) axes.

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```



Esquema Stopper

- Clase Stopper
 - Público:
 - Parámetros configurables de la lectura láser
 - Constructor
 - Crea un publisher del topic `cmd_vel/input/teleop` con mensajes tipo `geometry_msgs/Twist`
 - Crea un subscirber del topic `/scan` con mensajes del tipo `sensor_msgs/LaserScan`
 - Método `startMoving`
 - Implementa un bucle cerrado: mientras puede avanzar (dependiendo de la lectura del sensor) llama a la función `MoveForward`.
 - Privado:
 - Manejador del nodo
 - Publisher y Subscriber
 - Método `MoveForward`
 - Método `callback` para manejar lectura de sensor suscrito.



Stopper.h

```
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"

class Stopper {
public:
    // Tunable parameters
    const static double FORWARD_SPEED = 0.5;
    const static double MIN_SCAN_ANGLE = -30.0/180*M_PI;
    const static double MAX_SCAN_ANGLE = +30.0/180*M_PI;
    const static float MIN_DIST_FROM_OBSTACLE = 0.5; // Should be smaller
    than sensor_msgs::LaserScan::range_max

    Stopper();
    void startMoving();

private:
    ros::NodeHandle node;
    ros::Publisher commandPub; // Publisher to the robot's velocity command
    topic
    ros::Subscriber laserSub; // Subscriber to the robot's laser scan topic
    bool keepMoving; // Indicates whether the robot should continue moving

    void moveForward();
    void scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan);
};
```




Stopper.cpp (1)

```
#include "Stopper.h"
#include "geometry_msgs/Twist.h"

Stopper::Stopper()
{
    keepMoving = true;

    // Advertise a new publisher for the robot's velocity command topic
    commandPub = node.advertise<geometry_msgs::Twist>("
/cmd_vel_mux/input/teleop", 10);

    // Subscribe to the simulated robot's laser scan topic
    laserSub = node.subscribe("scan", 1, &Stopper::scanCallback, this);
}

// Send a velocity command
void Stopper::moveForward() {
    geometry_msgs::Twist msg; // The default constructor will set all
    commands to 0
    msg.linear.x = FORWARD_SPEED_MPS;
    commandPub.publish(msg);
}
```



Stopper.cpp (2)

```
// Process the incoming laser scan message
void Stopper::scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan)
{
    bool isObstacleInFront = false;

    // Find the closest range between the defined minimum and maximum angles
    int minIndex = ceil((MIN_SCAN_ANGLE - scan->angle_min) / scan->angle_increment);
    int maxIndex = floor((MAX_SCAN_ANGLE - scan->angle_min) / scan->angle_increment);

    for (int currIndex = minIndex + 1; currIndex <= maxIndex; currIndex++) {
        if (scan->ranges[currIndex] < MIN_DIST_FROM_OBSTACLE) {
            isObstacleInFront = true;
            break;
        }
    }

    if (isObstacleInFront) {
        ROS_INFO("Stop!");
        keepMoving = false;
    }
}
```



Stopper.cpp (3)

```
void Stopper::startMoving()
{
    ros::Rate rate(10);
    ROS_INFO("Start moving");

    // Keep spinning loop until user presses Ctrl+C or the robot got too
    close to an obstacle
    while (ros::ok() && keepMoving) {
        moveForward();
        ros::spinOnce(); // Need to call this function often to allow ROS to
        process incoming messages
        rate.sleep();
    }
}
```



run_stopper.cpp

```
#include "Stopper.h"

int main(int argc, char **argv) {
    // Initiate new ROS node named "stopper"
    ros::init(argc, argv, "stopper");

    // Create new stopper object
    Stopper stopper;

    // Start the movement
    stopper.startMoving();

    return 0;
};
```




- Editar el fichero CMakeLists.txt (líneas rojas):

```
cmake_minimum_required(VERSION 2.8.3)
Project(wander_bot)

...

## Declare a cpp executable
add_executable(stopper src/Stopper.cpp src/run_stopper.cpp)

## Specify libraries to link a library or executable target against
target_link_libraries(stopper ${catkin_LIBRARIES})
```

- Ejecutar

```
$ cd ~/sesion2
$ catkin_make
```

- No olvidar usar este comando cuando se crea un nuevo package

```
$ source devel/setup.sh
```



- herramienta para lanzar fácilmente múltiples nodos ROS
 - local o via SSH
 - asignar valores a parámetros del *Parameter Server*
- Toma como entrada uno o más ficheros de configuración XML (con extensión **.launch**), especificando:
 - parámetros a asignar
 - nodos a lanzar
- Si se usa **roslaunch** no hay que ejecutar **roscore**



- Copiar el fichero stopper.launch (descargarlo de PRADO) en el directorio launch (si no existe crearlo) del paquete

```
<launch>
  <param name="/use_sim_time" value="true" />
  <!-- Launch turtle bot world -->
  <include file="$(find turtlebot_gazebo)/launch/turtlebot_world.launch"/>

  <!-- Launch stopper node -->
  <node name="stopper" pkg="wander_bot" type="stopper" output="screen"/>
</launch>
```

Este launch lanza Gazebo y stopper node:

- To run the launch file:

```
$ roslaunch wander_bot stopper.launch
```

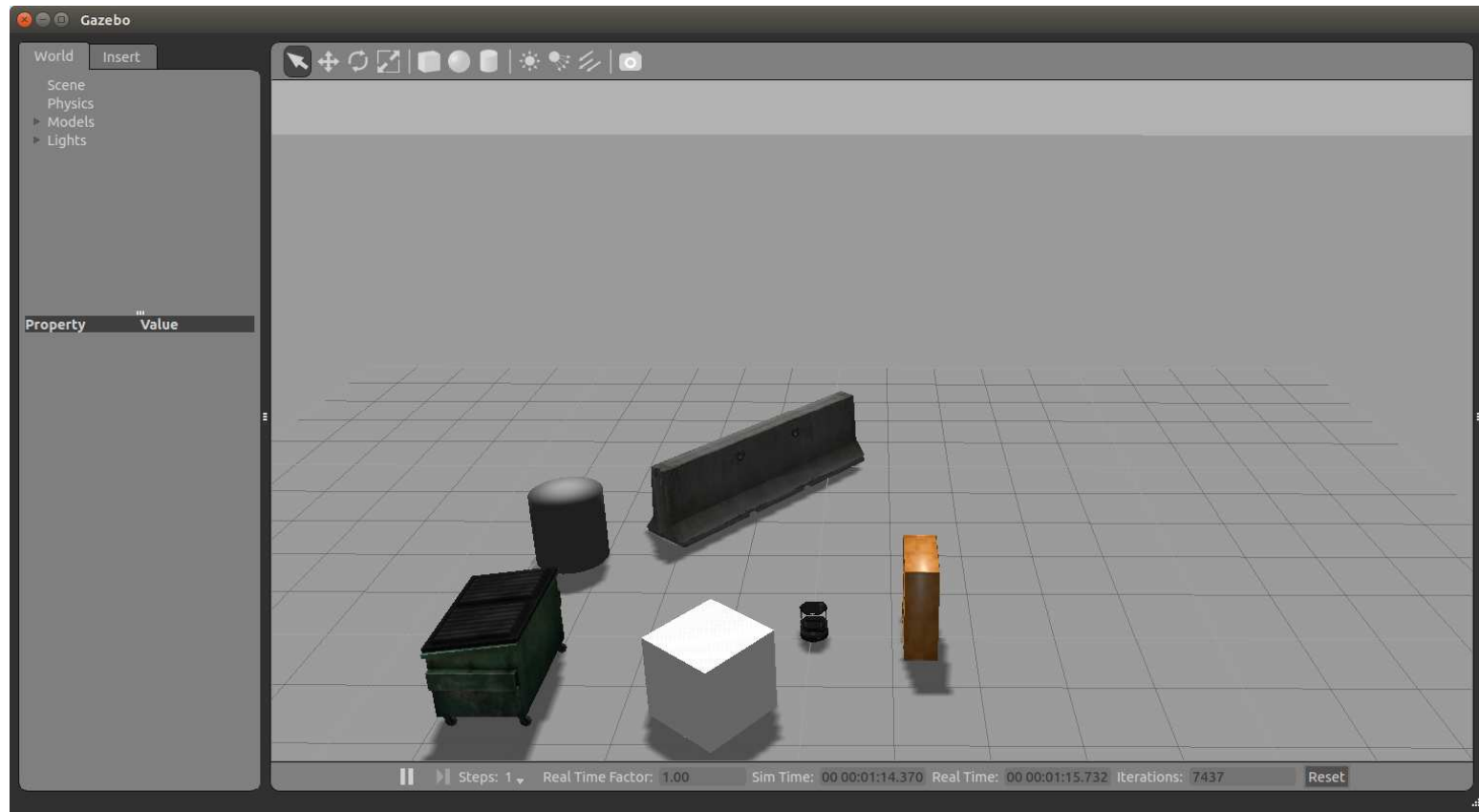
- Si no funciona, ejecutar cada nodo por separado con rosrn.

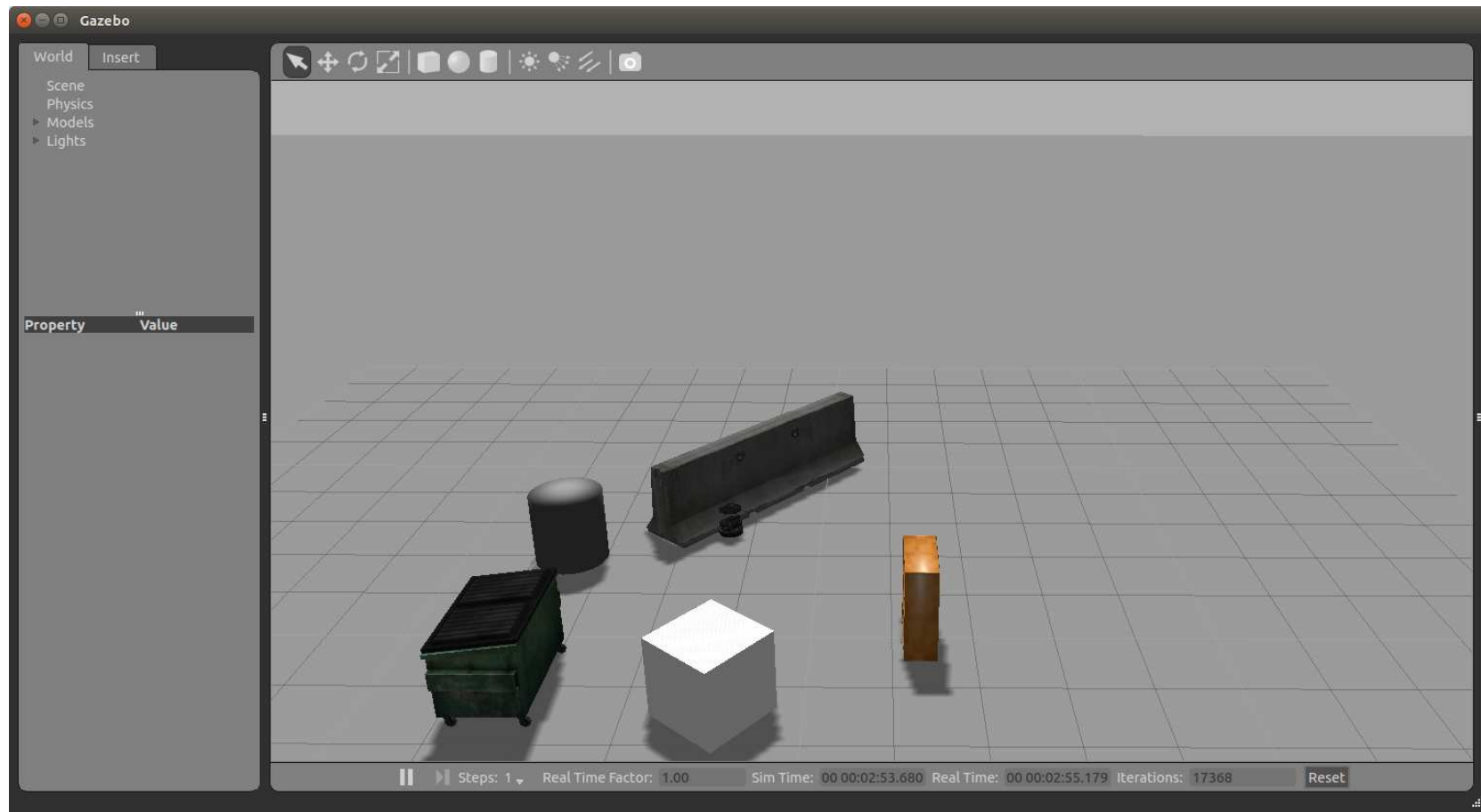


- Normalmente las librerías cliente de ROS usan el reloj del sistema como “wall-clock”.
- Cuando se ejecuta una simulación, es deseable a veces hacer que el sistema use un reloj simulado, con lo qque se puede acelerar, retardar o ir paso a paso sobre el tiempo percibido del sistema.
- Para dar soporte a esto, las librerías ROS pueden “escuchar” el topic `/clock` usado para publicar “tiempo de simulación”.
- Para este propósito, poner el parámetro `/use_sim_time` a true *antes de que se inicialice el nodo*.



Turtlebot Initial Position







- La Práctica 1 tiene dos entregas:
 - Entrega1: Implementar un algoritmo de navegación aleatoria. Ayudará basarse en *Stopper*.
 - Ver la descripción en el material de la práctica.
 - Valoración de esta entrega: 50% de la nota de la Práctica1.
 - La Entrega2 se describirá en la siguiente sesión.
- Entrega: 17 de Marzo, hasta las 23:55.