Introducción a la Robótica Móvil

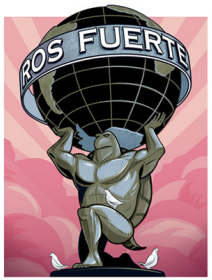
Primer cuatrimestre de 2018

Departamento de Computación - FCEyN - UBA

Introducción a ROS - clase 5

Robot Operating System

ROS: Robot Operating System



- → Framework open-source para el desarrollo de software para robots
- → Colección de herramientas, bilbiotecas (librerías) y convenciones para la creación de comportamientos robustos y complejos.
- → Incentiva el desarrollo colaborativo generando un “ecosistema” de soluciones a diversos problemas en Robótica.

ROS: Robot Operating System

Recursos

- **Web**: http://www.ros.org/
- **Blog**: http://www.ros.org/news/
- **Documentación**: http://wiki.ros.org/

Versiones

- **SO sugeridos: Ubuntu 14 ó Ubuntu 16**
- **Versiones sugeridas: Indigo ó Kinetic**

Filosofía de ROS

→ **Modular y Peer-to-Peer**

Los sistemas basados en ROS están constituidos por módulos que se comunican a través del envío y recepción de mensajes.

→ **Multilenguaje**

Los módulos de ROS pueden escribirse en cualquier lenguaje que posea una librería cliente (**client library**). Existen librerías cliente para **C++**, **Python**, LISP, Java, MATLAB, y más.

→ **Delgado**

ROS permite desarrollar librerías propias stand-alone y luego adaptarlas para ROS de manera que sean capaces de entender y enviar mensajes ROS.

→ **Basado en herramientas**

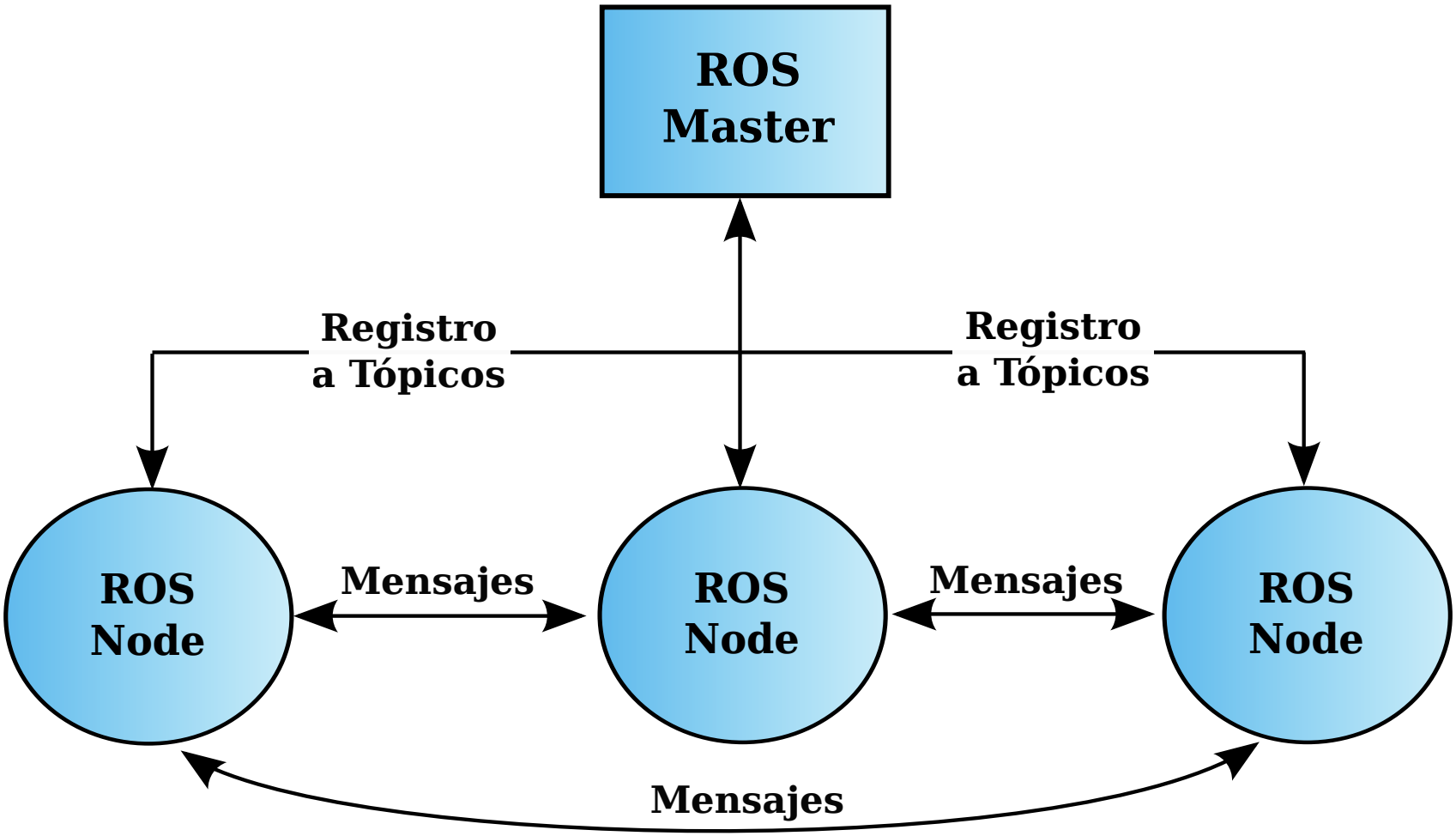
Tareas de visualización, registro (logging), inspección de información sensorial, comunicación con simuladores, etc.. son realizadas por medio de numerosas herramientas disponibles.

→ **Gratuito y open-source**

Componentes principales

1) **Infraestructura de comunicaciones:**
   - → Pasaje de mensajes.
   - → Llamadas a procedimientos remotos (RPC).
   - → Sistema distribuido de parámetros.
   - → Grabación y reproducción de mensajes (datasets).
2) **Características específicas para aplicaciones en robótica:**
   - → Conjunto de mensajes estandarizados.
   - → Librería geométrica: The Transform Library (tf).
   - → Lenguaje de descripción: Unified Robot Description Format.
   - → Estimación de pose, localización y navegación.
3) **Conjunto de herramientas de desarrollo, visualización y diagnóstico:**
   - → Herramientas de línea de comandos.
   - → RViz: visualización 3D de gran cantidad de tipos de información provista por sensores.
   - → RQt: entorno de desarrollo para interfaces gráficas.
   - → Interfaces para distintos simuladores (Gazebo, V-Rep).

Comunicación



Master

ROS Master

Provee información de conexión entre nodos de manera que puedan transmitir mensajes entre ellos.

- → Los nodos se reportan al **Master**, comunicando a cuáles tópicos publicarán mensajes y a qué tópicos se suscriben para recibir mensajes.
- → Una vez enlazados los nodos por medio del **Master**, forman una conexión directa *peer-to-peer* entre ellos.

Nodos y Tópicos

Nodos

- **Procesos** de propósito único. Ej: drivers de sensores, drivers de actuadores, procesamiento de imágenes, localización, etc.
- Compilados, ejecutados y manejados de manera individual.
- Escritos utilizando una librería cliente.
- Publican y se suscriben a tópicos.

Tópicos

- Representan un canal de comunicación entre nodos, conformado por un nombre y el tipo de mensajes que serán transmitidos. Ej: la información de un sensor laser podría ser enviada por el tópico “scan”.
- Los canales son anónimos, los nodos no conocen quién publica en el tópico.
- Modelo publicador / suscriptor: en general relación 1 a N.

Mensajes

- Estructura de datos compuesta por uno o varios campos tipados.
- Un campo puede ser de tipo primitivo (integer, float, boolean, etc.) o de tipo compuesto.
- Por ejemplo, geometry_msgs/Twist es utilizado para expresar comandos de velocidad:

geometry_msgs/Twist

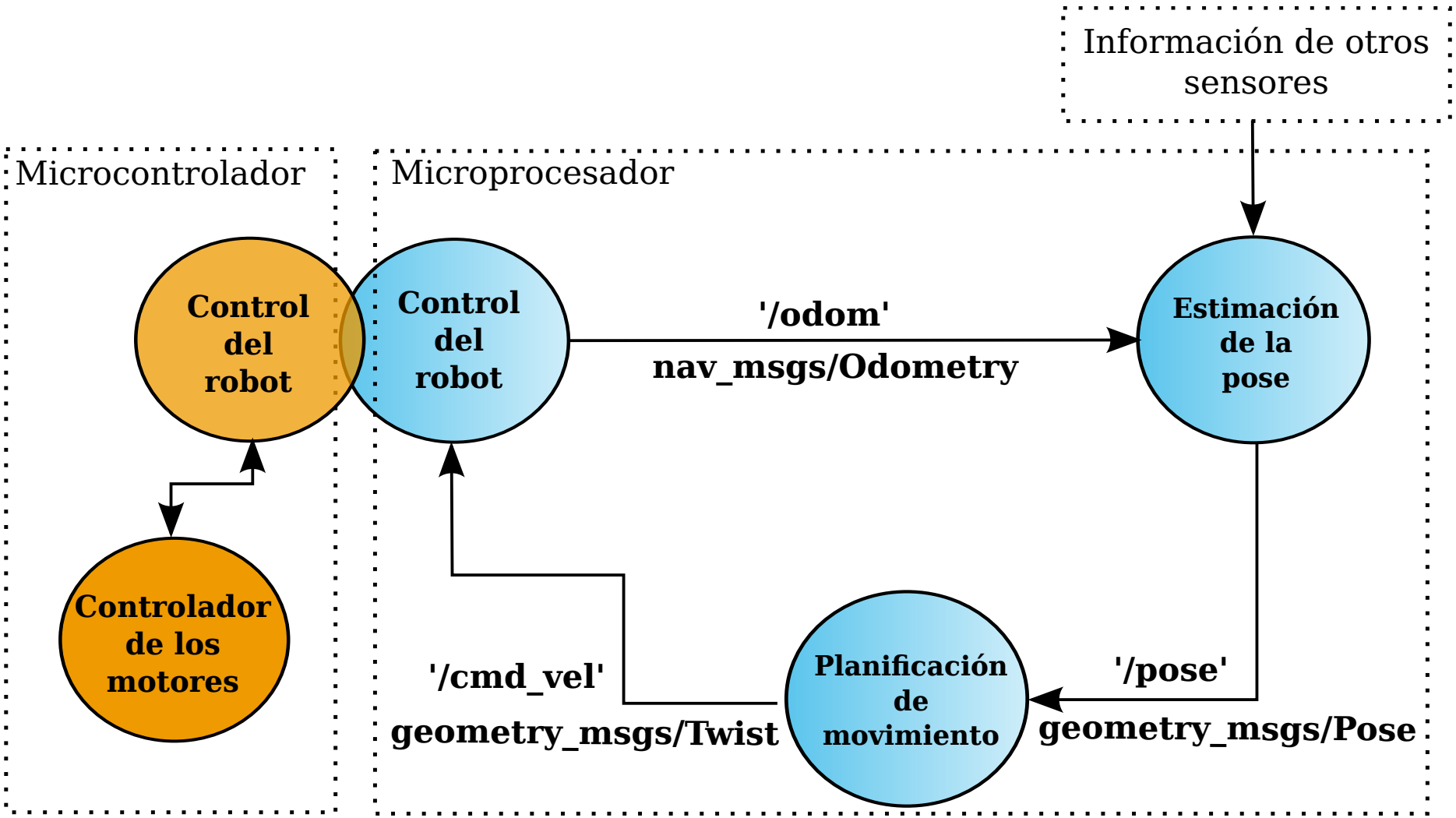
Vector3 linear
Vector3 angular

- Y a su vez Vector3 es un mensaje compuesto de:

geometry_msgs/Vector3

float64 x
float64 y
float64 z

Nodos y Mensajes



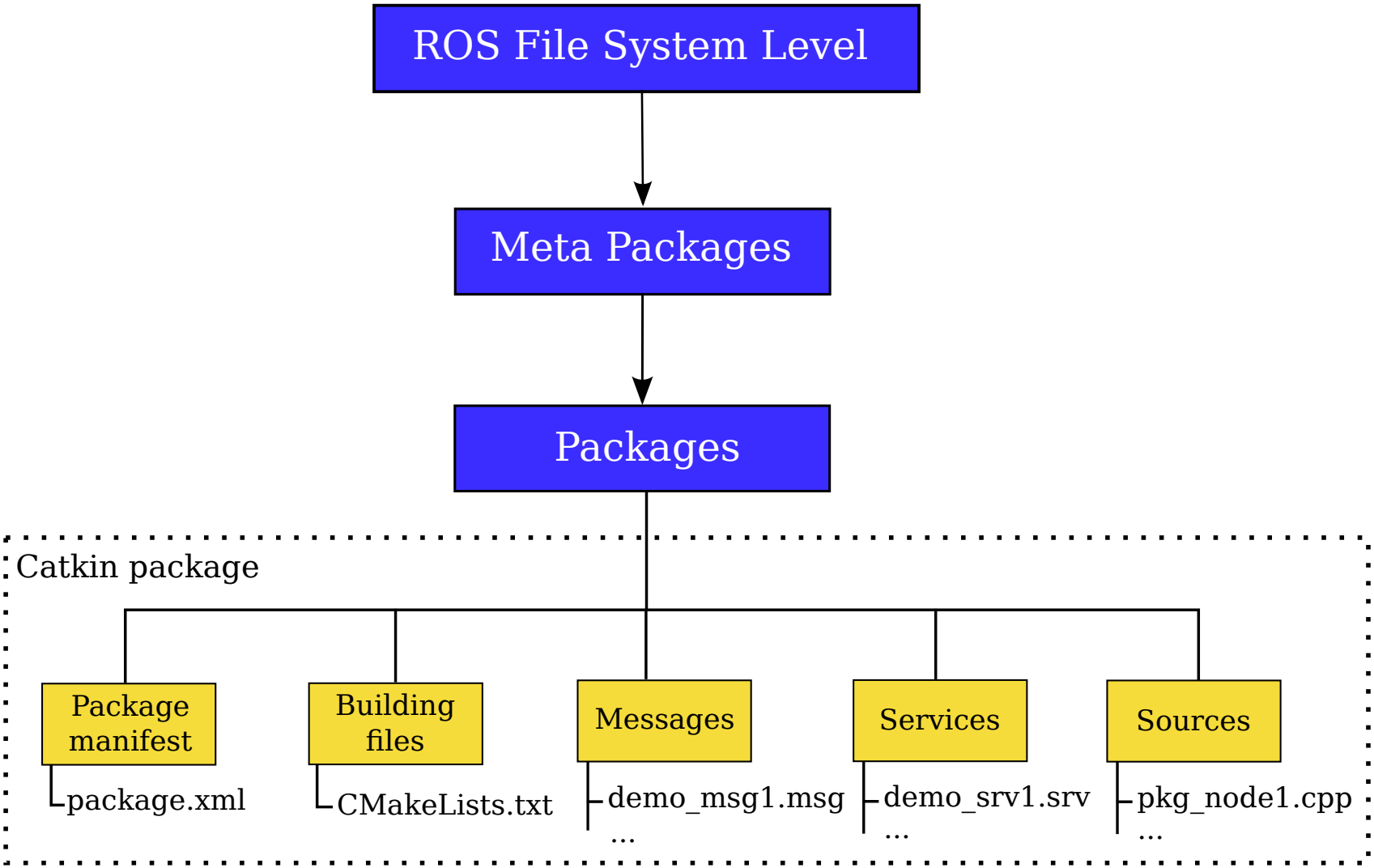
¿Cómo se implementa un nodo publicador?

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

int main(int argc, char **argv)
{
  // inicializacion del nodo, notificando el nombre
  ros::init(argc, argv, "nodo_publicador");

  // provee el medio para comunicarnos
  ros::NodeHandle handle;

  // publicaremos por el topico '/cmd_vel'
  ros::Publisher twists_pub =
    handle.advertise<geometry_msgs::Twist>("/cmd_vel", 1);

  // preparamos el mensaje
  geometry_msgs::Twist twist;
  twist.linear.x = 1;

  ros::Rate loop_rate(10); // frecuencia en Hz

  while ( ros::ok() ) {
    twist_pub.publish( twist ); // publicamos el Twist

    loop_rate.sleep(); // frecuencia requerida al ciclo
  }

  return 0;
}
```

¿Cómo se implementa un nodo suscriptor?

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

// los mensajes llegan como punteros
void twistCallback(const geometry_msgs::Twist::ConstPtr& msg)
{
  ROS_INFO("Velocidad lineal en X: [%d]", msg->linear.x);
}

int main(int argc, char **argv)
{
  // el nodo se inicializa, notifica su nombre
  ros::init(argc, argv, "nodo_suscriptor");

  ros::NodeHandle handle;

  // suscribe a 'cmd_vel' y especifica el callback
  ros::Subscriber sub =
    handle.subscribe("/cmd_vel", 1, twistCallback);

  // funcion bloqueante, ROS procesa mensajes
  ros::spin();

  // sale del spin() cuando el sistema se "apaga"

  return 0;
}
```

ROS File System



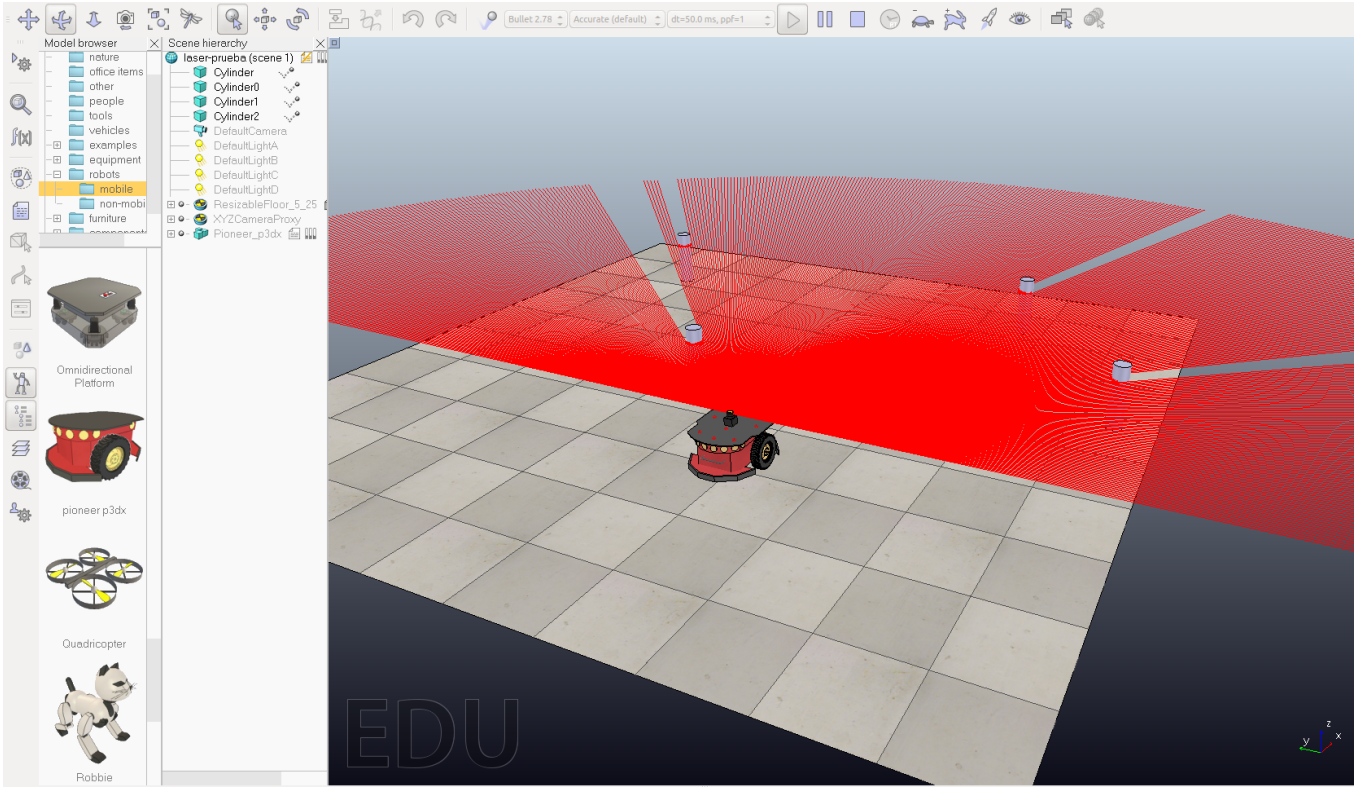
Demostración: ROS TurtleSim

→ http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

Algunas herramientas

`roscore`	inicializa el sistema y levanta el ROS Master.
`rosnode list`	lista los nodos existentes en el sistema.
`rosnode info`	lista tópicos a los cuales se publica y se suscribe un nodo.
`rosrun` **paq nodo**	permite ejecutar un nodo de un determinado paquete. Ej: `rosrun turtlesim turtlesim_node`
`rostopic list`	lista los tópicos activos.
`rostopic echo`	imprime por pantalla la información que esta siendo transmitida por un tópico.
`rostopic pub`	publica mensajes en un determinado tópico. Ej: `rostopic pub -r 10 /turtle1/cmd_vel geometry_msgs/Twist '[1.0,0.0,0.0]' '[0.0,0.0,1]'`
`rqt_graph`	genera un gráfico de los nodos y tópicos del sistema.

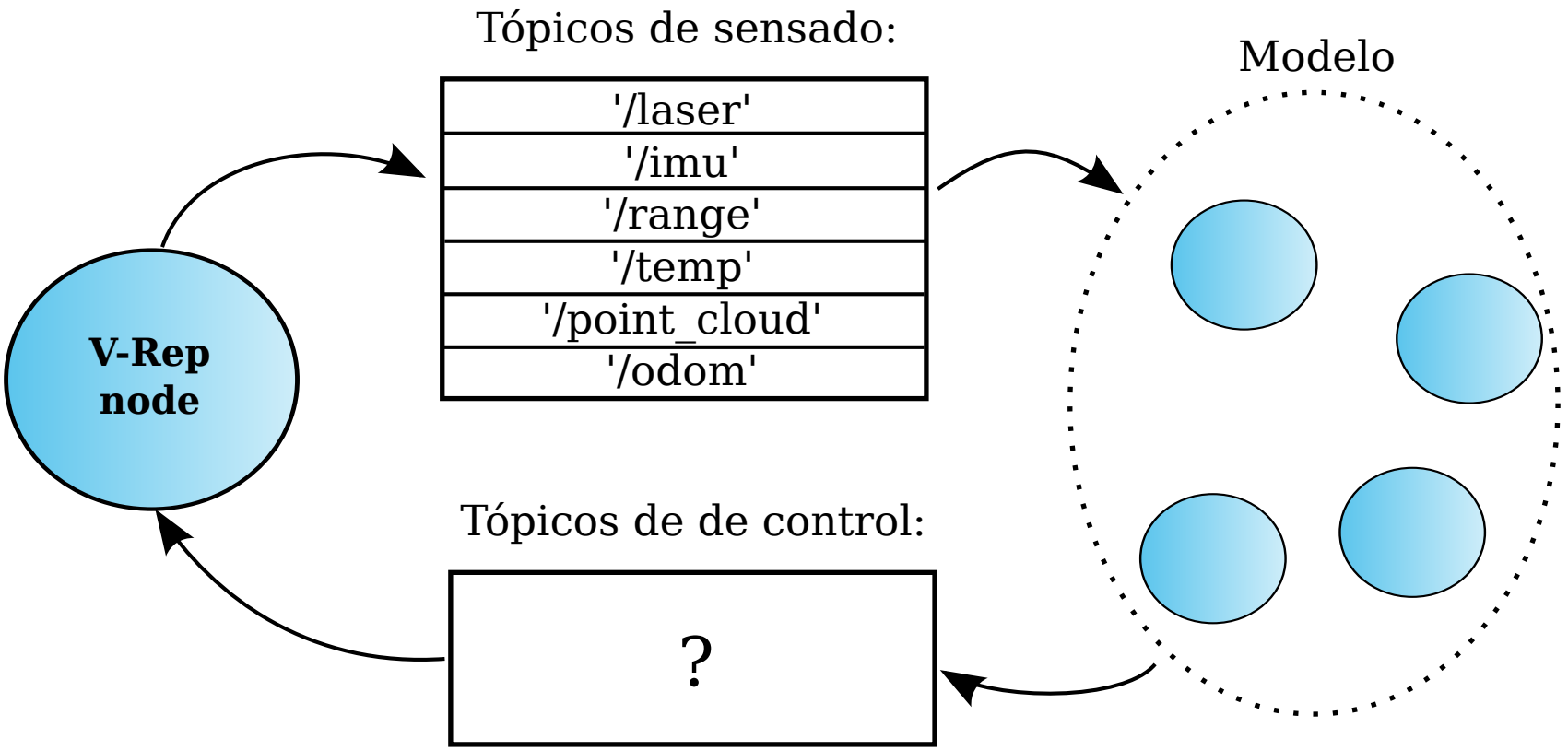
V-Rep: Simulador



http://www.coppeliarobotics.com/

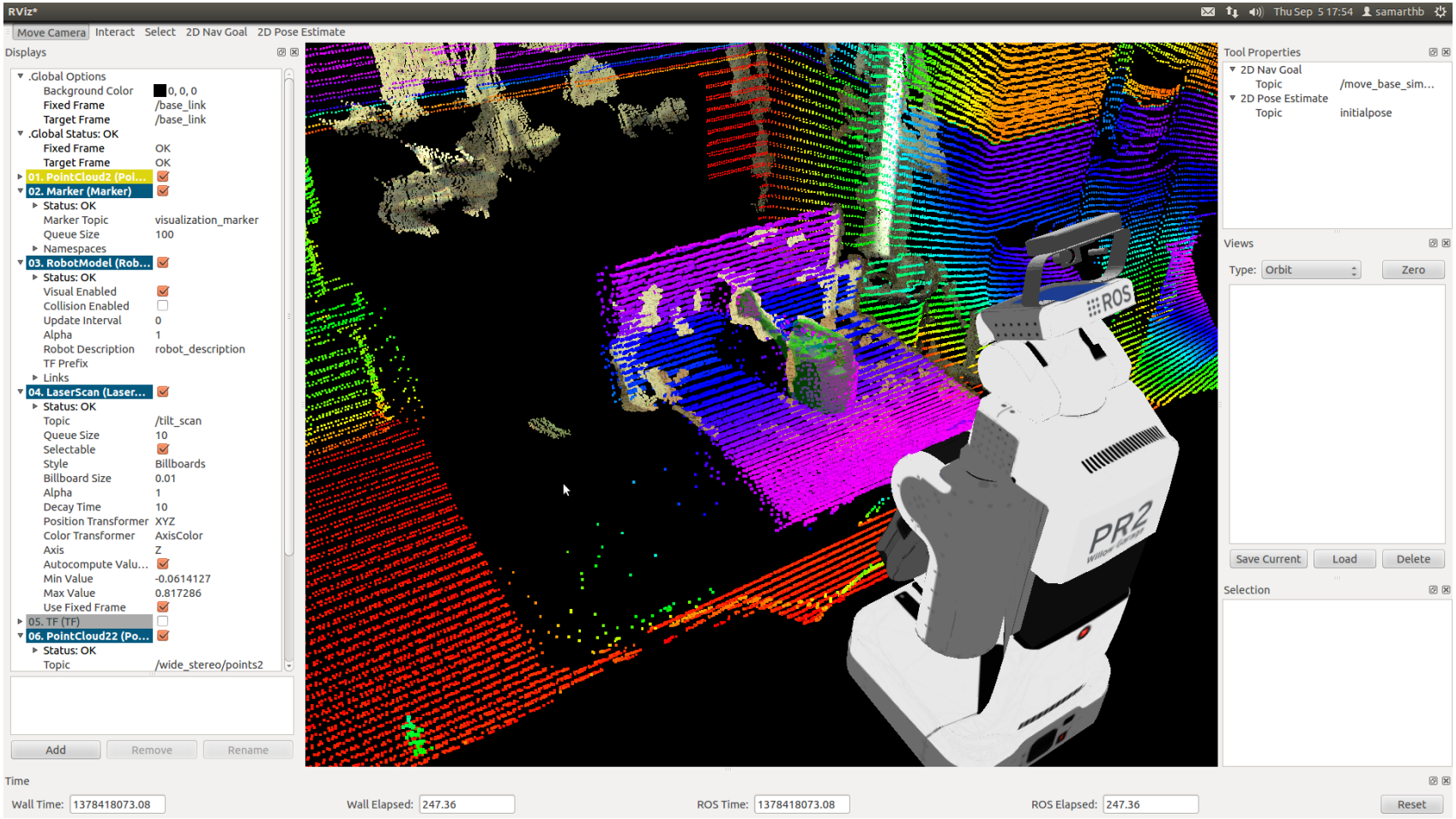
- → Permite crear escenarios estáticos y dinámicos.
- → Gran cantidad de **sensores** y robots listos para **simular**.
- → Interfaces remotas para diferentes entornos de desarrollo (ej: **ROS**).

Interfaz con ROS



¿Y los ticks del encoder?
Generalmente se resuelven a nivel de microcontrolador, pero nosotros los publicaremos y resolveremos la odometria a nivel de ROS.

RViz: Visualizador de ROS



→ Entorno gráfico que nos permitirá visualizar mensajes de manera sencilla.

→ Nos permite visualizar la forma en que el robot 'experimenta' el mundo.

¡Manos a la obra!

¡Bajar el enunciado del Taller 5 de la página de la materia!