uc3m | Universidad Carlos III de Madrid

Robots Móviles

Software: Introducción tutorial ROS

Gráficos de elaboración propia.

Jonathan Crespo Herrero

Contenido

- Definición de ROS
- Ventajas/inconvenientes
- Arquitectura
- Tutoriales

¿Qué es ROS?

- Robot Operating System (ROS). Es un framework muy extendido con el que se puede desarrollar software para robots.

- Es una colección de herramientas, librerías y convenios que intentan **simplificar** las tareas de crear robots complejos y robustos.

Contenido

- Definición de ROS
- **Ventajas/inconvenientes**
- Arquitectura
- Tutoriales

Ventajas de ROS

- Simplificación, reusabilidad, software libre.
- Permite centrarse en un problema concreto. Divide y vencerás! **Robótica colaborativa**.
- Se encuentra bastante extendido.
- Multilenguaje (C++, python, Java, Octave, Lisp, RxLab)
- Vídeo: http://www.youtube.com/watch?feature=player_embedded&v=PGaXiLZD2KQ

Desventajas de ROS

- Librerías muy heterogéneas, cada autor tiene su estilo.
- **Integración** puede ser costosa (Leer mucha documentación, depurar código, posibles errores).
- Curva de aprendizaje un poco lenta.
- Obsolescencia (las librerías cambian constantemente).

Ejemplo

- Queremos que un robot vaya al frigorífico y nos traiga una cerveza.

- Necesario:

  - Mapeo, navegación y localización.

  - Reconocimiento de objetos (frigorífico/cerveza).

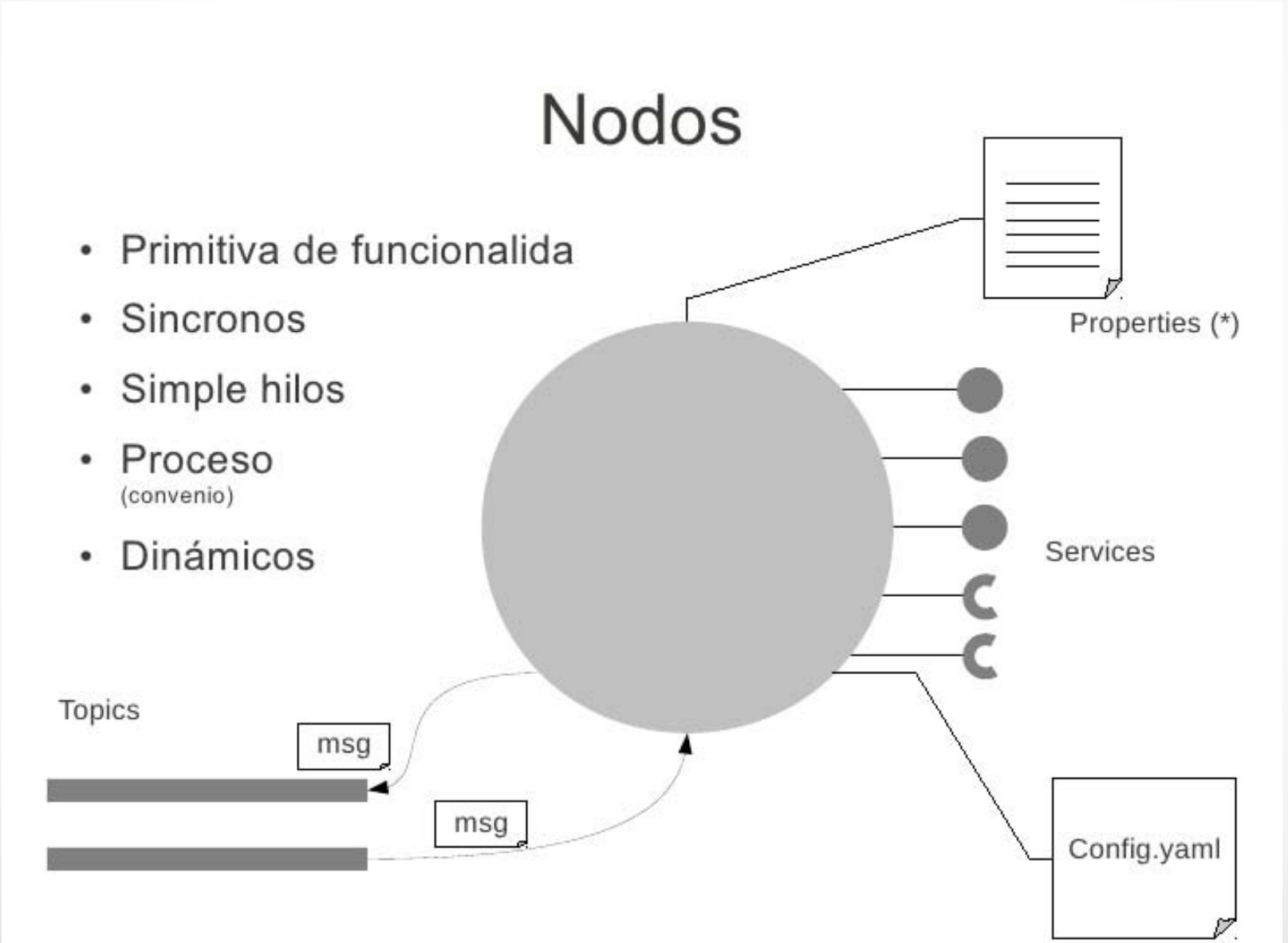
  - Reconocimiento de caras (que la lleve a quien la ha pedido).

  - Cinemática, grasping y manipulación de objetos.

Contenido

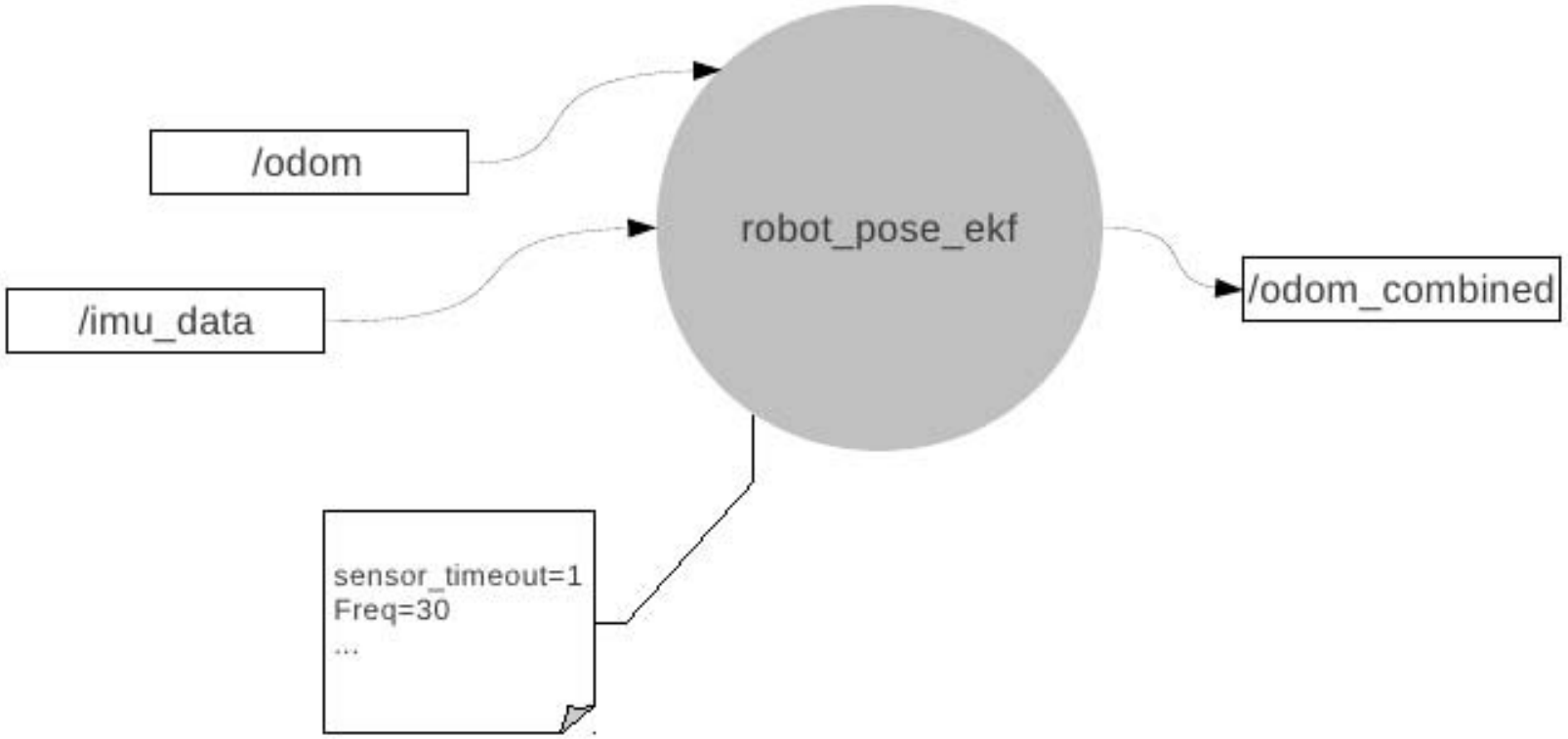
- Definición de ROS
- Ventajas/inconvenientes
- **Arquitectura**
- Tutoriales

Arquitectura de ROS



Ejemplo de “capacidad” Localización mediante EKF

- robot_pose_efk



Contenido

- Definición de ROS
- Ventajas/inconvenientes
- Arquitectura
- **Tutoriales**

Tutoriales de ROS

- Disponibles en http://wiki.ros.org/ROS/Tutorials
- Aprendizaje y manejo de los comandos de ros: roscd, rospack, roscreate-pkg,...
- Entendimiento de la arquitectura nodos/topics/servicios...
- Aprendizaje a empezar a escribir nuestros nodos

Comandos básicos

- roscd nombre_paquete
- rospack.
- roscreate-pkg
  rospack help
- rosbuild/catkin
- Configuración de variables ROS_MASTER_URI y ROS_HOSTNAME

Primeros pasos.

- Lancemos el roscore
- Herramientas útiles: rostopic, rosnode, rosparam
- Lancemos un nodo: rosrun turtlesim turtlesim_node
- Volvemos a lanzarlo: rosrun turtlesim turtlesim_node __name:=my_turtle

Tráfico de topics

- rostopic info → vemos el tipo, publicadores y suscriptores
- Lanzemos la “teleoperación”

  rosrun turtlesim turtle_teleop_key
- Qué sucede cuándo pulsamos las teclas? Mensajes publicados.

  rostopic echo /nombre_topic
- Probemos a lanzar los topics desde la consola
-

Tráfico de topics

- Para lanzar un topic desde la consola necesito el nombre del topic, el tipo de mensaje y saber la estructura del mensaje!!

  rostopic pub /nombre_topic tipo_topic valor

  valor: '{nombreCampo1: valor1, nombreCampo2: valor2}'

  se puede poner un ratio (-r 1) para que publique constantemente.

- rosmsg show tipo_mensaje

  (Localizar el archivo del mensaje)

Visualización de mensajes entre nodos

- Herramienta rqt_graph

  rosrun rqt_graph rqt_graph

- Ejemplo: Volver a lanzar el echo del topic de la velocidad y la publicación periódica. Entonces ver de nuevo el grafo.

- Velocidad de transmisión de mensajes:
  rostopic hz /nombre_topic
  Probar con la posición.

Servicios ROS

- En un servicio ROS, un nodo manda una petición a otro y espera una respuesta.

- rosservice
  rosservice call clear → llamada a servicio

- Cómo ver el tipo del servicio
  rosservice type spawn | rossrv show

- Llamada a servicio con valores de entrada
  rosservice call spawn 2 2 0.2 “Donatello”

Parámetros ROS

- Los parámetros son valores, típicamente de configuración, que emplean los nodos. Están almacenados en el ROS Master.

- rosparam

  rosparam get/set nombre_parámetro [valor]

- Ejemplo: Cambiamos parámetro color y ejecutamos el servicio clear.

Roslaunch

- El comando roslaunch inicia la serie de nodos que estén definidos en el archivo tipo launch

```
<launch>

  <group ns="turtlesim1">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <group ns="turtlesim2">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <node pkg="turtlesim" name="mimic" type="mimic">
    <remap from="input" to="turtlesim1/turtle1"/>
    <remap from="output" to="turtlesim2/turtle1"/>
  </node>

</launch>
```

Escribir un mensaje/servicio

- Archivos de texto con tipos de dato. Los servicios tienen entrada/respuesta.

- Dentro de la carpeta msg. Modificación de CMakeList.txt

Escribir un nodo que publica un topic

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Rate loop_rate(10);
  int count = 0;

  while (ros::ok())
  {
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

Escribir un nodo que recibe un topic

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
  ros::spin();

  return 0;
}
```

Nodo que implementa un servicio. Servidor.

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"

bool add(beginner_tutorials::AddTwoInts::Request  &req,
         beginner_tutorials::AddTwoInts::Response &res)
{
  res.sum = req.a + req.b;
  ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
  ROS_INFO("sending back response: [%ld]", (long int)res.sum);
  return true;
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_server");
  ros::NodeHandle n;

  ros::ServiceServer service = n.advertiseService("add_two_ints", add);
  ROS_INFO("Ready to add two ints.");
  ros::spin();

  return 0;
}
```

Nodo que implementa un servicio. Cliente.

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_client");
  if (argc != 3)
  {
    ROS_INFO("usage: add_two_ints_client X Y");
    return 1;
  }
  ros::NodeHandle n;
  ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");
  beginner_tutorials::AddTwoInts srv;
  srv.request.a = atoll(argv[1]);
  srv.request.b = atoll(argv[2]);
  if (client.call(srv))
  {
    ROS_INFO("Sum: %ld", (long int)srv.response.sum);
  }
  else
  {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }
  return 0;
}
```

Ejemplo de rviz con el Turtlebot

- Rviz es una herramienta que ayuda al ingeniero a identificar en tiempo real ciertos fallos.
- Se puede configurar y guardar dicha configuración para mostrar lo que interesa
- Muestra los sistemas de referencia de cualquier componente del robot o del entorno definida en el sistema, mediante los topics existentes.