Integración de ROS a la plataforma Butiá

Ignacio Escudero, Rodrigo Quinta, Andrés Tipoldi

Junio 2014

Agenda

- Objetivo
- Introducción a ROS
- Integrar ROS con Butiá
  - ¿Cómo?
  - Modelo URDF
  - Implementación Nodo/Driver
    - Demo

Objetivo

Integrar el robot Butiá a ROS brindando la posibilidad de controlarlo usando esta plataforma.

Objetivo

ROS

- Plataforma OpenSource de desarrollo de software de robótica.
- Comunicación entre procesos
- Abstracción del hardware
- Bibliotecas de alto nivel
  - Manejo de sistemas de coordenadas
  - Procesamiento de imágenes

La integración de ROS a la plataforma Butiá brindaría la posibilidad de implementar comportamientos y arquitecturas de control aprovechando las herramientas que brinda el sistema.

ROS

- ROS es un “meta-operating system”
  - Licencia de código abierto BSD
- Provee una forma de comunicación entre procesos por medio de mensajes en red.
- Brinda una abstracción de hardware, device drivers, librerías, visualizadores y herramientas.
  - Ayuda a los desarrolladores de software a crear aplicaciones para robots.

ROS

- Runtime “graph” de ROS:
  - Red peer-to-peer de procesos poco acoplados, que usan la infraestructura de comunicación de ROS.

- ROS implementa diferentes formas de comunicación:
  - Comunicacion sincronica (RPC-style) sobre servicios
  - Comunicación asíncrona basada en tópicos
  - Almacenamiento de información en un “Parameter Server”.

Conceptos

- **Nodos**
  - Procesos encargados de un comportamiento.
  - Un sistema de control de un robot puede estar integrado por varios nodos.
  - Forman un grafo en el que se comunican entre ellos utilizando los topics y servicios.
  - Por lo general un nodo se encarga de una única función del robot (path planning, mover motores, etc).
  - Proporciona modularidad al sistema:
    - Mejor manejo de fallas
    - Individualidad de los nodos (exponen una API).

Conceptos

- **ROS Master**
  - Provee una forma de registro y búsqueda para el resto de la red.
  - Permite establecer conexiones entre los nodos (similar a DNS)
    - Los nodos no se comunican mediante el Master, la comunicación es entre nodos, el master simplemente establece esta comunicación.
  - Almacena la informacion para el registro a Tópicos y Servicios y actualiza a otros nodos sobre el status de los demás.
  - Provee el Parameter Server.

Conceptos

- **Parameter Server**
  - Servidor compartido que almacena parámetros que son consultados y almacenados por los nodos en runtime.
  - Utilizado principalmente para almacenar parámetros de configuración.
  - Implementado utilizando XMLRPC.
  - Por convención los parámetros son nombrados siguiendo una jerarquía para evitar colisiones de nombres.

Conceptos

- **Mensajes**
  - Los nodos se comunican a través de mensajes.
  - Estructura de datos con atributos tipados.
  - Existen varios tipos estándar predefinidos
  - Se pueden definir nuevos tipos.
  - Pueden tener una estructura anidada, un mensaje puede estar integrado por otros mensajes.
  - ROS genera una estructura de datos para los mensajes nuevos que contienen funciones estándar.

Conceptos

- **Tópicos**
  - Proporcionan un via de transporte con semántica publish / subscribe.
  - Los nodos envían mensajes a los tópicos y los nodos suscriptos a éste reciben una notificación.
  - Por lo general suscriptores y los que publican no se conocen.
  - Son un bus de mensajes y cualquiera se puede conectar para enviar o recibir mensajes

Conceptos

- **Servicios**
  - Proporcionan comunicacion one-to-one, Request/Reply.
  - Los nodos envían mensajes de solicitud a los nodos que proveen servicios.
  - Los nodos que proveen servicios responden con un mensaje reply.
  - Llamado a servicio es bloqueante.

Integrar ROS con Butiá

- **¿Que hay que hacer?**
  - Crear un Robot Stack: Butia Stack
- **Componentes**:
  - Driver
  - Nodo
  - Modelo
  - Inicialización

Integrar ROS con Butiá

- **Driver:** butia_driver
  - Driver para el robot.
  - Podemos Usar PyBot Client
- **Nodo:** butia_node
  - Wrapper para el driver.
  - API de ROS exponiendo los servicios (y tópicos) del robot.

Integrar ROS con Butiá

- **Modelo:** butia_description
  - URDF del robot
  - Descripción física: ubicación de sensores, actuadores
  - Sensores móviles en Butia:
    - Modelo fijo con configuración típica
    - Lego NXT – Lego Digital Designer

Integrar ROS con Butiá

- **Inicialización:** butia_bringup
  - Scripts de startup
  - Se invocan funciones de inicialización

MODELO

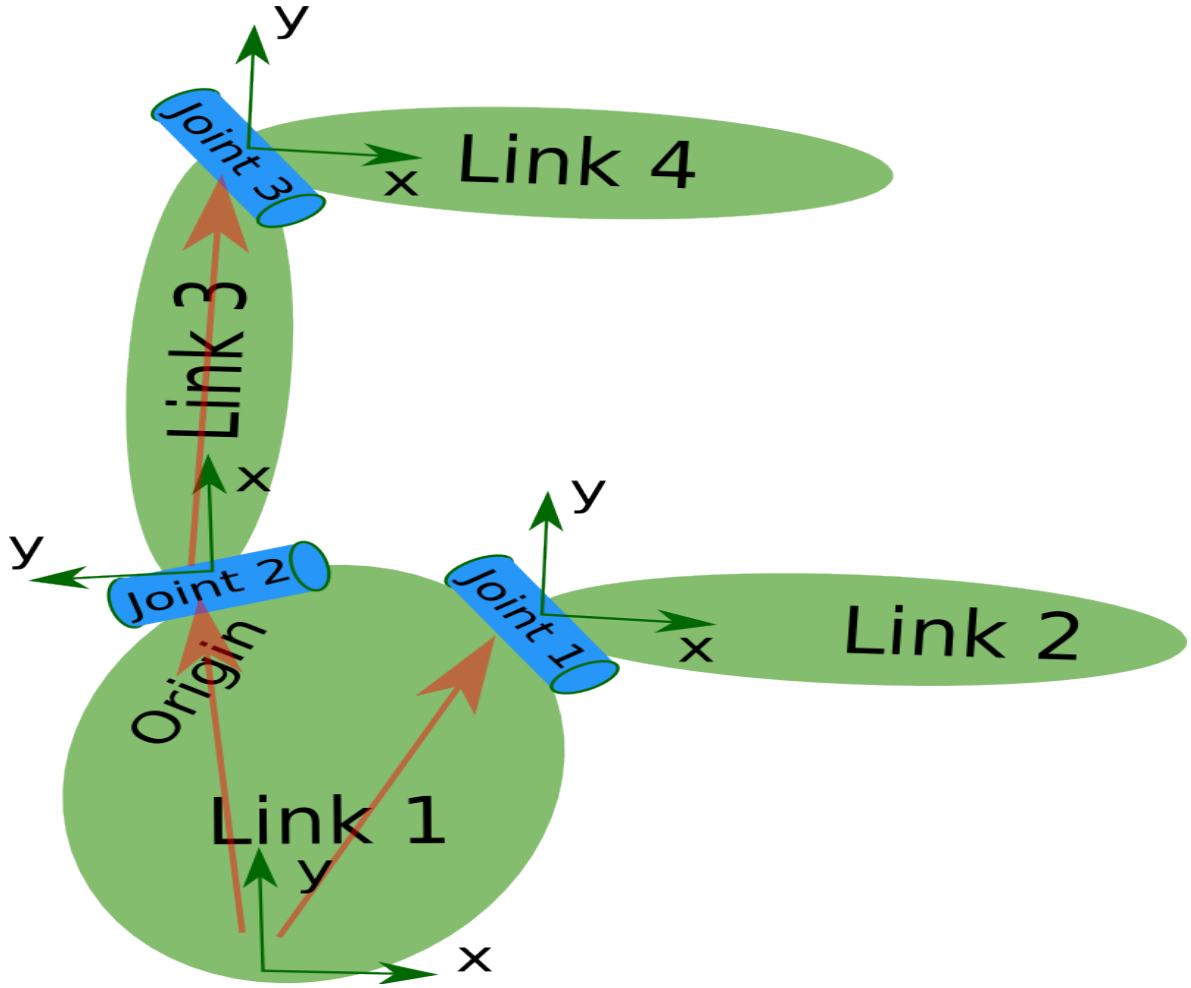
URDF-Unified Robot Description Format

- Especificación en XML de la descripción del Robot
  - Soporta 3 estructuras (Links, Joint y Robot(modelo))
  - Proporciona:
    - Descripción de la dinámica.
    - Representacion visual.
    - Modelado de colisiones.

URDF-Componentes

- Modelo:
  - Describe las propiedades cinemáticas y dinámicas de la estructura del robot.
  - Se compone por un conjunto de links y joints.

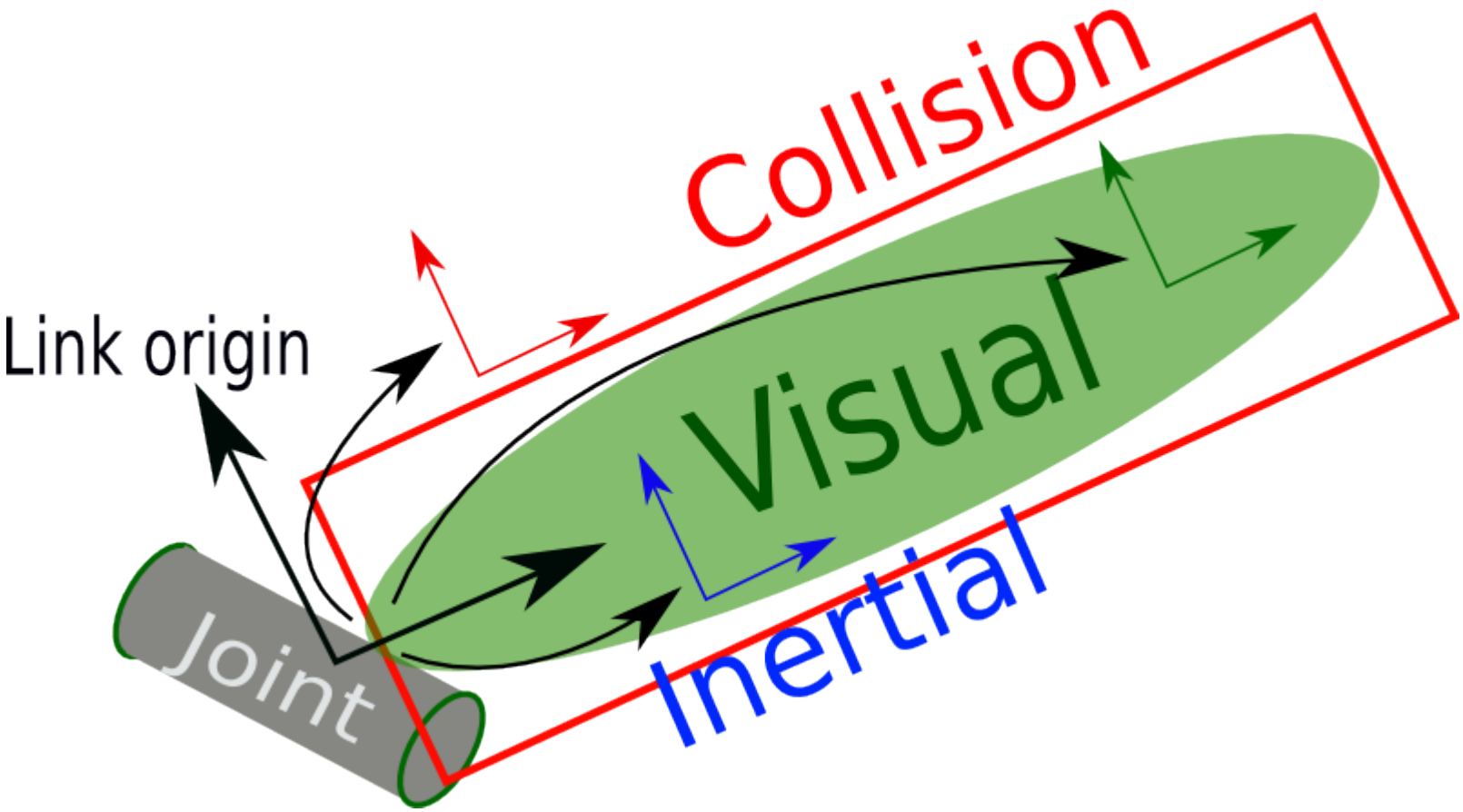
Modelo


y
Joint 3
x
Link 4
Link 3
x
y
Joint 2
Origin
y
Joint 1
x
Link 2
Link 1
y
x

URDF-Componentes

- Link:
  - Describe las propiedades cinemáticas y dinámicas de un link.
  - Describe un cuerpo rígido con inercia.
  - Propiedades visuales.
  - Propiedades de colisión.

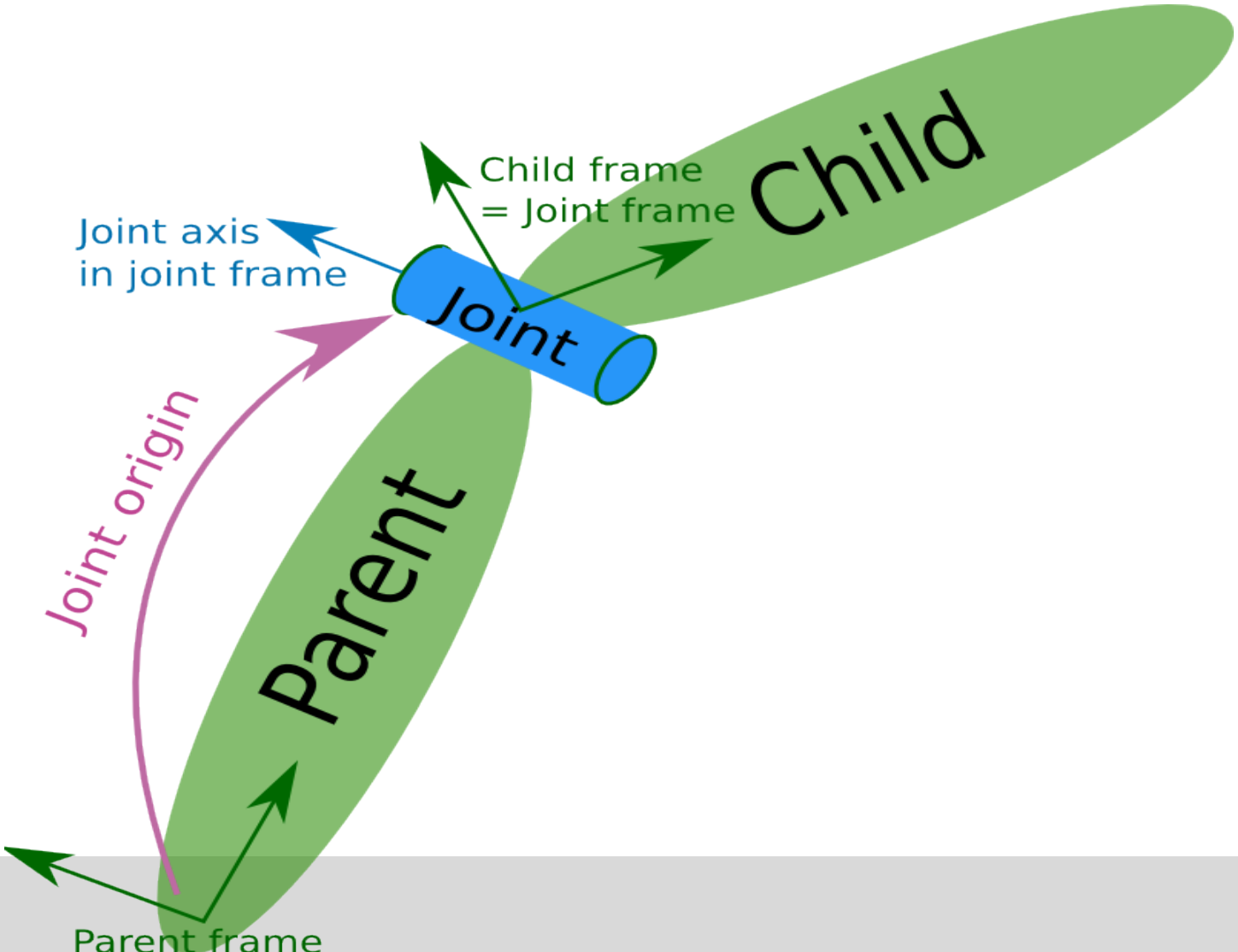
Link



URDF-Componentes

- Joint:
  - Describe las propiedades cinemáticas y dinámicas de una unión.
  - Describe los límites de seguridad de la unión. (Safety Limits)

Joint


Child
Child frame
= Joint frame
Joint axis
in joint frame
Joint
Joint origin
Parent
Parent frame

URDF-Extensiones

- Existen dos elementos que son extensiones del modelo URDF:
  - Transmission.
  - Gazebo.

URDF-Extensiones

- Transmission: Describe la unión entre los joints y los actuadores.

- Elementos:
  - Joint y el actuador que se relacionan,
  - Tipo de transmision.

URDF-Extensiones

- Gazebo: Describe propiedades para ser utilizadas con el simulador Gazebo.
- El simulador muestra una imagen 3D del robot con movimiento.

URDF-Otros

- Ademas existen 2 elementos que no son utilizados en la práctica.
  - Sensor: Projecto Abandonado.
  - Model State: Work in Progress.

URDF-Sensor

- Sensor: incluido en el Dom pero nunca utilizado.
- El proyecto fue abandonado pero se encuentra disponible si alguien lo quiere extender.
- Su objetivo es describir los sensores.

URDF-model_state

- Model_state: Describe el estado del modelo para un cierto tiempo.
- Es un proyecto en desarrollo.
- Actualmente la representación de las configuraciones del robot se hacen mediante SRDF (Semantic Robot Description File)

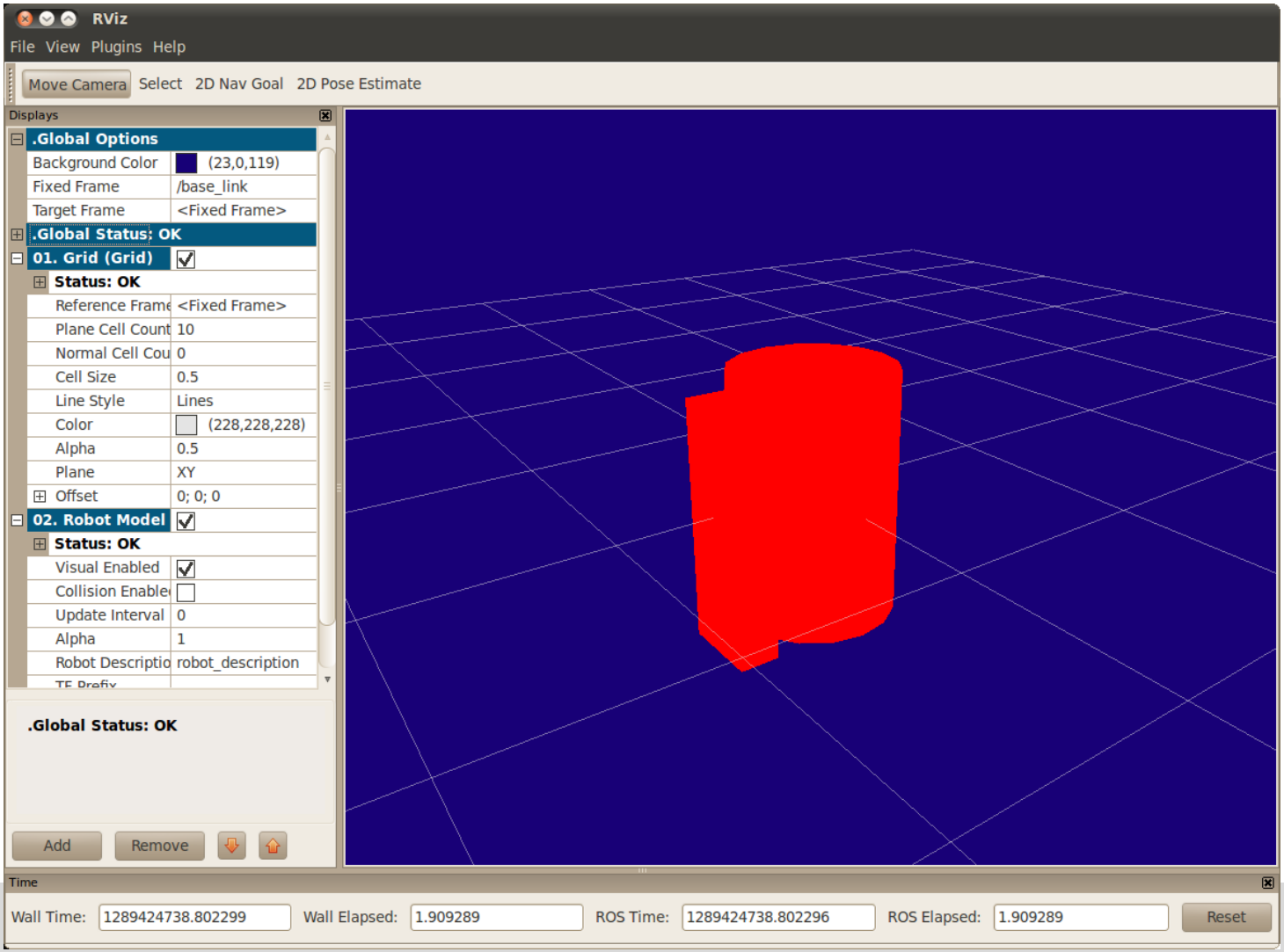
URDF-Xacro

- Para simplificar los archivos xml del URDF se utiliza Xacro.
- Xacro: es un macro lenguaje xml.
- ROS permite generar archivos URDF a partir de los archivos Xacro.
  - rosrun xacro xacro.py model.xacro > model.urdf

URDF-Otros

- Herramientas incluidas en ROS:
  - Graphiz: permite generar un pdf con el modelo.
  - Rviz: Permite ver el modelo y sus partes en 3D

Rviz


RViz
File View Plugins Help
Move Camera
Select
2D Nav Goal
2D Pose Estimate
Displays
.Global Options
Background Color (23,0,119)
Fixed Frame /base_link
Target Frame <Fixed Frame>
.Global Status: OK
01. Grid (Grid)
Status: OK
Reference Frame <Fixed Frame>
Plane Cell Count 10
Normal Cell Cou 0
Cell Size 0.5
Line Style Lines
Color (228,228,228)
Alpha 0.5
Plane XY
Offset 0; 0; 0
02. Robot Model
Status: OK
Visual Enabled
Collision Enable
Update Interval 0
Alpha 1
Robot Descriptio robot_description
.Global Status: OK
Add
Remove
Time
Wall Time: 1289424738.802299
Wall Elapsed: 1.909289
ROS Time: 1289424738.802296
ROS Elapsed: 1.909289
Reset

URDF-Publicar

- Robot State Publisher: Se encarga publicar la información del URDF.
- Ejecutado como Nodo o como Librería.
- Necesita de un archivo URDF cargada en el parameter server.

Modelado del Mundo

- El modelado del mundo se encuentra la librería tf Transform.
- Mantiene un conjunto de marcos de las posiciones de las partes del robot.
- Obtinene la informacion del URDF que fue publicada por el Robot State Publisher.

tf Transform

- Dos tareas principales:
  - Escuchar Transformaciones:
    - Almacena todos los marcos de coordenadas que recibe.
    - Consulta por cambios especificos.
  - Publicar Transformaciones:
    - Publica los marcos de coordenadas de las distintas partes del robot.

IMPLEMENTACIÓN

Implementación

- Driver
  - PyBot Client
    - Comunicación con PyBot Server
    - Usamos la API que ofrece

Implementación

- Nodo
  - Wrapper del Driver
  - Expone servicios de ROS
  - Publica en tópicos de ROS

Implementación

- **Servicios:** butia_ros_server.py
  - butia_get_gray
  - butia_get_distance
  - butia_get_button
  - butia_set_2_motor_speed

Implementación

- Definición de **Mensajes**
  - ButiaGetGray.srv
  - ButiaGetDistance.srv
  - ButiaGetButton.srv
  - ButiaSet2MotorSpeed.srv

Implementación

- **Consumir Servicios**: butia_ros_client.py
  - Parámetros
    - get_gray *a*
    - get_distance *a*
    - get_button *a*
    - set_2_motor_speed *a b c d*

Implementación

- **Tópicos**: butia_ros_server_topics.py
  - Publica los valores de los sensores
    - get_gray
    - get_distance
    - get_button
  - Se invoca definiendo
    - Nombre del tópico
    - Comando
    - Frecuencia

Implementación

- Inicializar un Nodo

```
rospy.init_node('butia_ros_server')
```

- Definir un Servicio

```
s = rospy.Service('butia_get_gray', ButiaGetGray,
handle_get_gray)
```

- Tipo de Mensaje: ButiaGetGray.srv

```
int64 a
---
int64 value
```

Implementación

- Invocar un Servicio

```
rospy.wait_for_service('butia_get_gray')
service = rospy.ServiceProxy('butia_get_gray',
ButiaGetGray)
value = service(a)
```

Implementación

- Definir un Tópico

```
pub = rospy.Publisher('butiaGray', String)
```

- Publicar en un Tópico

```
pub.publish(value)
```

- Suscribirse a un Tópico

```
rospy.Subscriber('butiaGray', String, callback)

def callback(msg):
    print "Recibido %s"%msg.data
```

Demo