Introducción a Robotic Operating System

Actuación y cinemática

Martín Llofriu y Gonzalo Tejera
Facultad de Ingeniería :: Instituto de Computación

mina
network management | artificial intelligence

Contenido

- Servidor/consumidor
- El robot Butia
- El paquete butiaros
- Moviendo al robot
- Nodo piloto

Contenido

- Publicando odometría
- Transformando coordenadas
- Incluyendo la transformada de odometría
- Modelado URDF
- Robot state publisher
- Grabando odometría y sensado

Servidor/consumidor

Un servicio es ofrecido por un nodo

La entrada y salida del servicio (contrato) se especifica como un archivo de texto

```
int64 a
int64 b
---
int64 sum
```

Servidor/consumidor

Es necesario realizar ciertas modificaciones a los archivos que definen el paquete:

- package.xml:

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

- CmakeLists.txt:

```
find_package(catkin REQUIRED
        COMPONENTS
        roscpp
        rospy
        std_msgs
        message_generation)
```

Práctico: servidor/consumidor

- Creando un servicio:
http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv#Creating_a_srv

- Creando un servidor y cliente:
http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29
http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28python%29

Práctico: servidor/consumidor

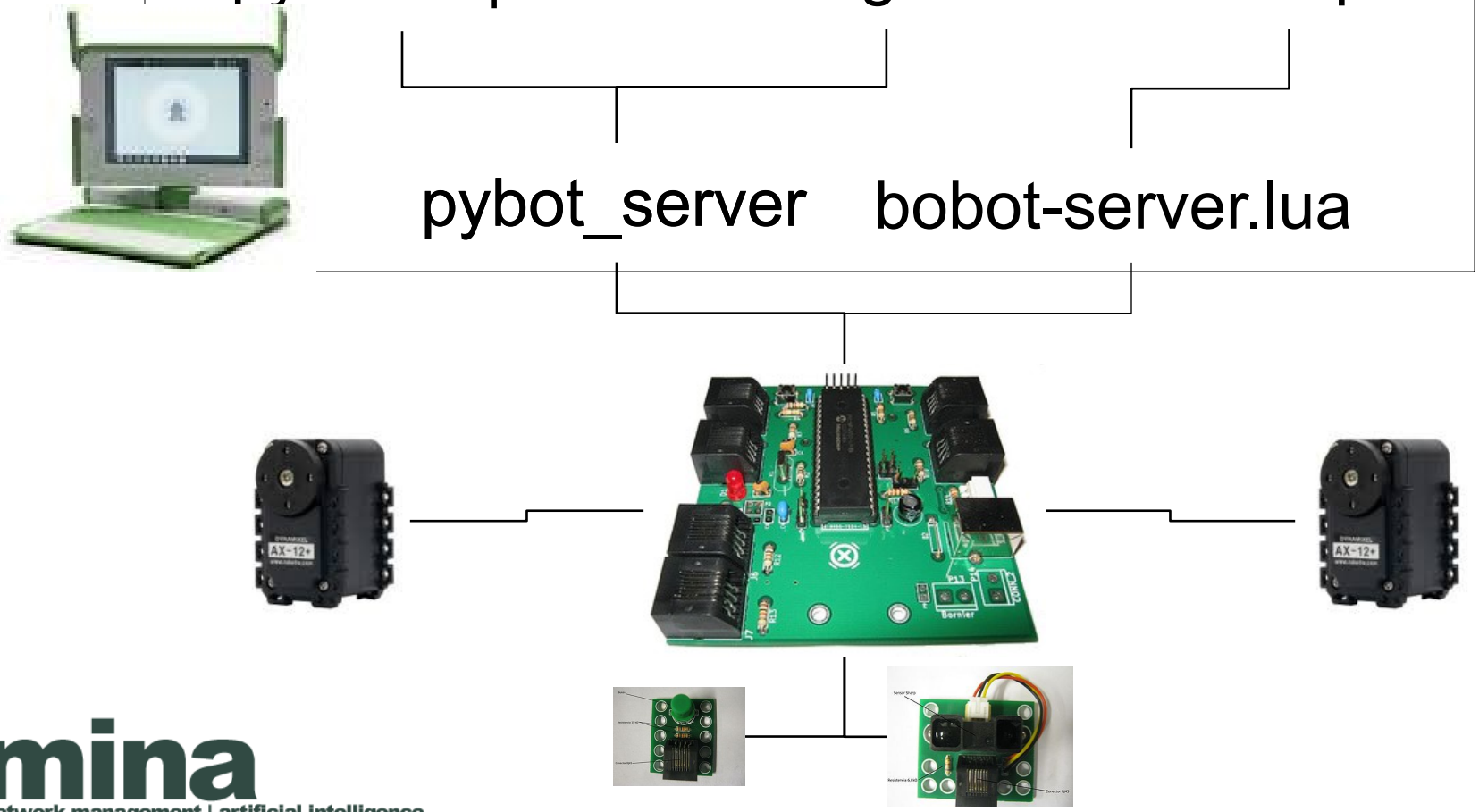
Consejos:

- El archivo de AddTwoInts.srv puede encontrarse en https://github.com/ros/ros_tutorials/blob/indigo-devel/rospy_tutorials/srv/AddTwoInts.srv
- Aunque se utilice python es necesario invocar catkin_make para generar los servicios

Robot Butiá

Recurso:
http://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/P%C3%A1gina_principal

Robot Butiá


python api
tortugarte
lua api
pybot_server
bobot-server.lua
AX-12+
AX-12+

Robot Butiá

- La placa USB4Butia se encarga de I/O con los sensores y motores
- Los servidores pybot y lua-server prestan servicios de consulta a sensores y envío de comandos a los motores
- Los sensores conectados son autodetectados

Robot Butiá + ROS

ROS + Paquete rosbutia

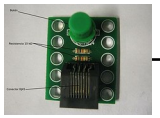
python api

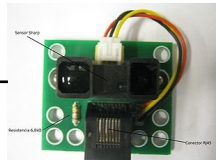
tortugarte

lua api

pybot_server

bobot-server.lua





mina
network management | artificial intelligence

El paquete rosbutia

Recurso:
http://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/Integracion_ros_butia

El paquete Butiá (ROS)

Características:

- Pasamanos de servicios expuestos por pybot
  - Consulta de sensores
  - Comandos motores
- Publicador genérico de tópicos
  - Se le pasa como argumentos el servicio pybot a consultar y la frequencia

El paquete Butiá (ROS)

Para ejecutar:

- roscore
- python pybot_server.py
- rosrun Butia butia_ros_server.py
- rosrun Butia butia_ros_server_topics.py Button 10 get_button 2

Práctico: Instalando rosbutia

Pasos a seguir:

- Hacer un usuario en github.com
- Entrar a https://github.com/mllofriu/butiaros y hacer fork del repositorio
- En el directorio src/ de nuestro workspace: git clone <url del nuevo repositorio>
- catkin_make

Práctico: usando rosbutia

Recurso:
http://www.fing.edu.uy/inco/proyectos/butia/mediawiki/index.php/Integracion_ros_butia

Pasos a seguir:

- Iniciar roscore, pybot y rosbutia
- Consultar el valor de un sensor usando el cliente
- Mover el robot
- Utilizar rostopic para comprobar el tópico publicado

Nodo Piloto

Existen formas estandar de publicar servicios de movimiento de un robot.

El tópico **cmd_vel** se asume relacionado a un servicio de movimiento del robot.

Ventajas:

- Se acopla a muchas otras bibliotecas de navegación
- Este mismo nodo puede encargarse de publicar odometría

Nodo Piloto

El tópico:

geometry_msgs/Vector3 linear

geometry_msgs/Vector3 angular

Práctico: Nodo Piloto

Recurso:
http://answers.ros.org/question/29706/twist-message-example-and-cmd_vel/

Es necesario convertir la velocidad lineal y angular en velocidades de rotación del robot.

Como convención:

- El eje *x* apunta hacia adelante del robot, *y* hacia la izquierda y *z* hacia arriba
- La rotación en el plano es *yaw* y corresponde a la componente *z* de la velocidad angular

Práctico: Nodo Piloto

Ecuaciones cinemáticas inversas

Publicando Odometría

Otra convención de ROS es la publicación de un tópico de odometría.

La idea básica es publicar un tópico con la posición y velocidad estimada del robot con respecto a un marco de referencia fijo.

Publicando Odometría

El mensaje odometría:

Header header

string child_frame_id

geometry_msgs/PoseWithCovariance pose

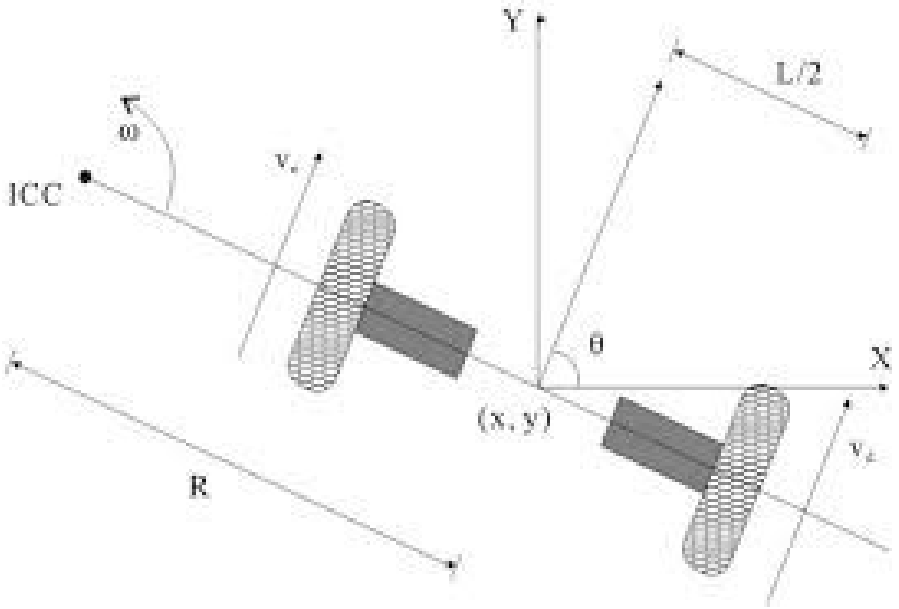
geometry_msgs/TwistWithCovariance twist

Tendremos que integrar los movimientos (forward kinematics) sabiendo las velocidades lineales y angulares en cada momento.

Forward Kinematics

El robot está caracterizado por:

- Su posición en el plano
  - x e y
- Su orientación
  - theta

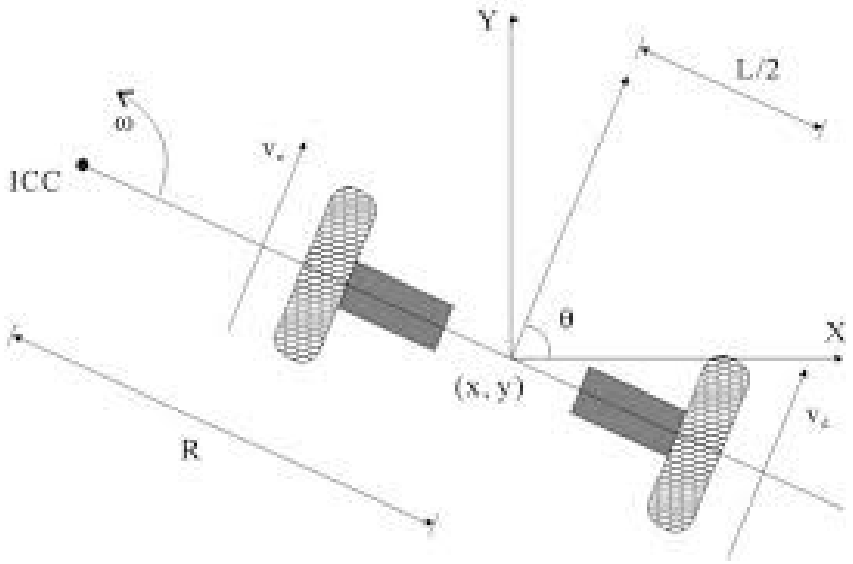


Forward Kinematics

Dx = vx * cos(theta) * dt

Dy = vx * sin(theta) * dt

Dtheta = vtheta * dt



Donde:

- *Dx* y *Dy* son cambios en *x* e *y*
- *dt* es el intervalo de tiempo transcurrido
- *vx* y *vtheta* son las velocidades en *x* y *theta* respectivamente (en el marco del robot)

Práctico: Publicando Odometría

Recurso:
http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom

Pasos a seguir:

- Agregar las dependencias a CmakeLists.txt
- Agregar un loop principal a pilot que integre el movimiento cada cierto intervalo
- Crear un mensaje odometry con esa información y publicarlo
- Saltearemos por ahora la parte de odom_trans

Práctico: Publicando Odometría

Consejos:

- Para dormir un intervalo determindo: http://wiki.ros.org/rospy/Overview/Time
- Para crear un quaternion: quaternion_about_axis(angle, (0,0,1))

Transformación de coordenadas

- El paquete tf nos permite transformar puntos en un marco de coordenadas a otro
- También incorpora la dimension temporal
- En base a transformadas entre dos links, tf concatena las transformaciones

Práctico: publicar una transformada

Recurso:
http://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20broadcaster%20%28Python%29

Pasos a seguir:

- Programar un nodo que publique una transformada constante entre dos frames cualesquiera
- Iniciar el nodo
- Iniciar rviz y desplegar información de tf

Práctico: transform listener

Este objeto es capaz que encontrar la transformada entre dos marcos de referencia cualesquiera

Luego, es posible usar esa transformada para transformar las coordenadas de un punto

Práctico: transform listener

Pasos a seguir:

- Crear un nodo con un transform listener
- Encontrar la transformada entre dos marcos
- Transformar un punto utilizando la transformada obtenida
- Imprimir las coordenadas del punto y corroborar con cálculos “a mano”

Recursos:

- Transform listener: http://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20listener%20%28Python%29

Incluyendo la transformada de odometría

Los nodos que publican odometría publican también una transformada con la posición del robot

Práctico: transformada odometría

Pasos a seguir:

- Volver al recurso de publicación de odometría http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom
- Identificar el código que publica la transformada
- Incluir un código equivalente en el nodo de butiaros

Modelado URDF

Los modelos Unified Robot Description Format (URDF) nos permiten modelar las relaciones entre joints y links, especificando estructura, medidas, masas y grados de libertad.

En base a esta estructura, podemos transformar datos en las coordenadas de un sensor a las coordenadas de otra parte del robot.

Archivos URDF

```
<?xml version="1.0" ?>
<robot name="dm3">
        <link name="front_base">
        ...
        </link>
        <link name="front_right_wheel">
        ...
        </link>
        <joint name="front_base_to_right_wheel"
type="continuous">
        ...
        </joint>
</robot>
```

Archivos URDF

```
<link name="front_base">
        <visual>
                <geometry>
                        <box size="0.24 0.10 0.05"/>
                </geometry>
                <material name="blue">
                        <color rgba="0 0 .8 1"/>
                </material>
        </visual>
        ...
</link>
```

Archivos URDF

```
<link name="front_base">
        <collision>
                <geometry>
                        <box size="0.24 0.10 0.05"/>
                </geometry>
        </collision>
</link>
```

Archivos URDF

```
<link name="front_base">
        <inertial>
                <origin rpy="0 0 0" xyz="0.0 0.0 0.0"/>
                <mass value="10"/>
                <inertia ixx="0.0104" ixy="0" ixz="0" iyy="0.05"
                  iyz="0" izz="0.05"/>
        </inertial>
</link>
```

Archivos URDF

```
<joint name="front_base_to_right_wheel" type="continuous">
        <parent link="front_base"/>
        <child link="front_right_wheel"/>
        <origin rpy="0 0 0" xyz="0.22 0 0"/>
        <limit effort="100" velocity="100"/>
        <joint_properties damping="0.0" friction="10"/>
</joint>
```

Práctico: crear un URDF para Butiá

Pasos a seguir:

- Tomar las medidas del robot
- Crear un link central que corresponda a la plataforma
- Crear dos links cilíndricos para las ruedas
- Crear dos joints continuos para las ruedas
- Crear un link para la cámara. Z se perpendicular al plano de proyecccion
- Crear un joint constante de la plataforma a las ruedas

Práctico: crear un URDF para Butiá

Para probar:

- Setear el parametro robot_description para que apunte al texto del archivo
  rosparam set –text-file=<full-path> robot_description
- Ejecutar rviz e incluir un modelo del robot

Robot State Publisher

El paquete robot_state_publisher nos permite utilizar esta información para mantener las matrices de transformación de coordenadas entre un link y el siguiente.

De esta forma, podemos transformar información desde cualquier par de coordenadas unidas por esta cadena de transformaciones.

Robot State Publisher

El paquete robot_state_publisher necesita de los estados de todos los joints.

Estos estados serán proporcionados por el paquete joint_state_publisher.

Practico: robot_state_publisher

Crear un archivo roslaunch que:

- Establezca el parametro robot_description
- Inicie el nodo joint_state_publisher del mismo paquete
- Inicie el nodo robot_state_publisher del mismo paquete

Recursos:

- joint_state_publisher: http://wiki.ros.org/joint_state_publisher
- robot_state_publisher: http://wiki.ros.org/robot_state_publisher

Práctico: Grabando odometría

Pasos a seguir:

- Programar un nodo que comande al robot a realizar un pequeño loop de movimiento
- Generar un archivo roslaunch que inicie el nodo piloto, el nodo de la cámara, el nodo de detección ar_pose y el nodo de loop
- Colocar algunas marcas en un circulo y poner al robot a moverse en el interior
- Grabar todos los tópicos obtenidos utilizando rosbag