



Prototyping (autonomous) Robots using Python and ROS

Simon Haller

08.11.2016, Innsbruck





ROS

Environment

URDF & TF

RoboCup Junior

Odometry and Obstacle Avoidance

OpenCV

Simulators

Further Topics

Current Robotic Frameworks

- ▶ Microsoft Robotics Studio
- ▶ NXJ (Open source Java environment for the Lego robot kit)
- ▶ TRS see <http://ulgrobotics.github.io/trs/>¹
- ▶ Orocos (c++ framework for component-based robot control software)
- ▶ Rock (the Robot Construction Kit)
- ▶ Orca (robot framework)
- ▶ CARMEN (robot simulator)
- ▶ OpenRTM (robotics technology middleware)
- ▶ and many more ...

¹If you use matlab in your classes this might be interesting

What is ROS?

- ▶ A software framework for developing robotic systems in a meta-operating system environment.
- ▶ The primary goal of ROS is to support code reuse in robotics research and development
- ▶ ROS was originally developed in 2007 at the Stanford Artificial Intelligence Laboratory and development continued at Willow Garage
- ▶ Managed by the Open Source Robotics Foundation

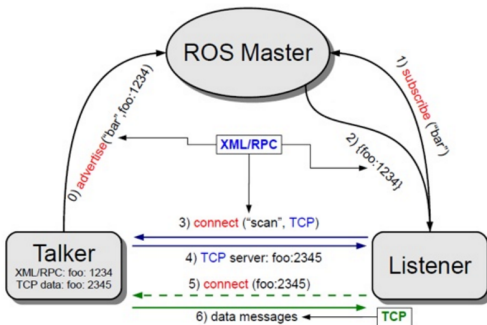
Features

- ▶ A collection of packaging, software building tools
- ▶ An architecture for distributed inter-process/machine
- ▶ Communication and configuration
- ▶ Development tools for system runtime and data analysis
- ▶ Open-source under permissive BSD licenses (ros core libraries)
- ▶ A language-independent architecture (c++, python, lisp, java, and more)
- ▶ A scalable platform (ARM CPUS to Xeon Clusters)

ROS Core Concepts

- ▶ Master(roscore): Provides naming and registration services to the rest of the nodes in the system.
Enables ROS nodes to locate one another and provide parameter services
- ▶ Package: A virtual directory holding one or more executables (nodes)
- ▶ Messages: Defined in msg files. Basic data types
- ▶ Node: An agent communicating with ROS and other nodes via messages
- ▶ Topics (publish / subscribe) using typed messages
- ▶ Services: Request / Response, synchronize functionality, remote computation
- ▶ Bag: Bags are a format for recording and playback of messages

ROS Architecture





ROS

Environment

URDF & TF

RoboCup Junior

Odometry and Obstacle Avoidance

OpenCV

Simulators

Further Topics



Environment

use .bashrc functions

```
function ros_local() {  
    getmyip  
    source /opt/ros/indigo/setup.bash  
    export ROS_MASTER_URI=http://$MYIP:11311  
    echo "set ROS_MASTER_URI=$ROS_MASTER_URI"  
    export ROS_HOSTNAME=$MYIP  
    export ROS_IP=$MYIP  
    export MY_WORKSPACE=/home/c703101/iis_robot_sw  
    if [ -f $MY_WORKSPACE/devel/setup.bash ] ; then  
        echo "sourcing $MY_WORKSPACE/devel/setup.bash"  
        source $MY_WORKSPACE/iis_catkin_ws/devel/setup.bash  
    fi  
}
```

Environment

use .bashrc functions

```
function getmyip() {  
    MYIP=$(ip addr show eth0 | grep "inet " | awk '{print $2}' \  
        | sed 's%/.*$%%g')  
    if [ -z $MYIP ] ; then  
        MYIP=$(ip addr show wlan0 | grep "inet " | awk '{print $2}' \  
            | sed 's%/.*$%%g')  
    fi  
    if [ -z $MYIP ] ; then  
        MYIP=127.0.0.1  
    fi  
    echo "using IP: $MYIP"  
}
```



Environment

use bash history

```
HISTSIZE=20000000  
HISTFILESIZE=20000000
```

optimize commands

```
cat $HOME/.bash_history | awk '{n[$1 " " $2]++} END \\  
    {for(i in n) printf "%1i %s \n", n[i], i}' | sort -n
```

Exercise - Terminal

- ▶ open `.bashrc` and look at the defined functions
- ▶ load basic ros environment

```
# Terminal 1
```

```
cat .bashrc
```

```
ros_local
```

```
roscore
```

```
# Terminal 2
```

```
ros_local
```

```
rostopic list
```

```
rosservice list
```

```
rosparam list
```



ROS

Environment

URDF & TF

RoboCup Junior

Odometry and Obstacle Avoidance

OpenCV

Simulators

Further Topics

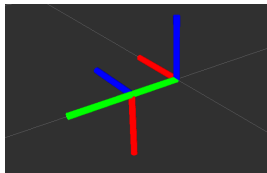
Unified Robot Description Format

Physical Representation → XML → ROS Tools

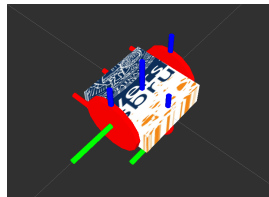
```
<robot name="basic">
  <link name="base_link" />
  <link name="right_wheel" />

  <joint name="base_to_right_wheel" type="fixed">
    <parent link="base_link"/>
    <child link="right_wheel"/>
    <origin xyz="0.0 0.16 0.0"
            rpy="0.0 1.57 0.0" />
  </joint>
</robot>
```

X – Red – Roll
Y – Green – Pitch
Z – Blue – Yaw



```
<?xml version="1.0"?>
<robot name="simple">
  <link name="base_link">
    <visual><geometry><box size="0.3 .3 .1"/></geometry></visual>
  </link>
  <link name="right_wheel">
    <visual>
      <geometry><cylinder length="0.02" radius="0.12"/></geometry>
      <origin rpy="1.57 0 0" xyz="0 0 0"/>
    </visual>
  </link>
  <link name="left_wheel">
    <visual>
      <geometry><cylinder length="0.02" radius="0.12"/></geometry>
      <origin rpy="1.57 0 0" xyz="0 0 0"/>
    </visual>
  </link>
  <joint name="base_to_right_wheel" type="continuous">
    <parent link="base_link"/>
    <child link="right_wheel"/>
    <origin xyz="0 0.16 0"/>
    <axis xyz="0 1 0" />
  </joint>
  <joint name="base_to_left_wheel" type="continuous">
    <parent link="base_link"/>
    <child link="left_wheel"/>
    <origin xyz="0.0 -0.16 0.0" />
    <axis xyz="0 1 0" />
  </joint>
</robot>
```



URDF Joints

- ▶ **fixed** Joint which allows no motion.
- ▶ **continuous** Joint that can rotate around a single axis.
- ▶ **revolute** Rotate with upper and lower angle limit.
- ▶ **prismatic** Linearly shift along a single axis, with upper and lower position limits.
- ▶ **planar** Translation and rotation perpendicular to a plane.
- ▶ **floating** Allows full 6D translation and rotation.



URDF Texture, Collada, Collision

Texture

```
<material name="pattern">  
  <texture filename="package://myrobot/pics/logo.jpg"/>  
</material>
```

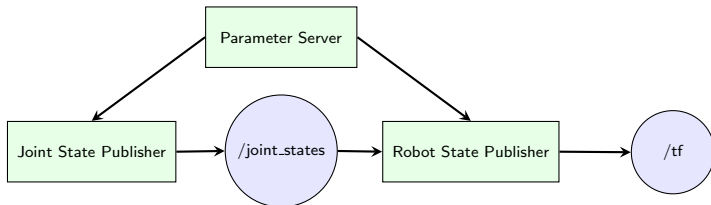
Collada

```
<geometry>  
  <mesh filename="package://myrobot/cad/link.dae" />  
</geometry>
```

Collision

```
<collision>  
  <geometry>  
    <mesh filename="package://myrobot/cad/link_simple.stl"/>  
  </geometry>  
</collision>
```

Parameter Server, Robot & Joint State Publisher



```
<launch>
  <arg name="gui" default="False" />
  <param name="robot_description" textfile="$(find myrobot)/urdf/simple.urdf" />
  <param name="use_gui" value="$(arg gui)"/>
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
</launch>
```



Tools: XACRO

```
<xacro:property name="wheel_size" value=".012" />
<xacro:property name="robot_width" value=".3" />
<xacro:macro name="wheel" params="prefix reflect">
  <link name="${prefix}_wheel">
    <visual>
      <geometry>
        <cylinder length="0.02" radius="${wheel_size}" />
      </geometry>
      <origin xyz="0 0 -${robot_width/2}" />
    </visual>
  </link>
  <joint name="base_to_${prefix}_wheel" type="fixed">
    <parent link="base_link"/>
    <child link="${prefix}_wheel"/>
    <origin xyz="0 ${reflect*robot_width} 0" />
  </joint>
</xacro:macro>
<xacro:wheel prefix="right" reflect="1" />
<xacro:wheel prefix="left" reflect="-1" />
```

Launch File Differences

```
<param name="robot_description" textfile="$(find myrobot)/urdf/simple.urdf" />

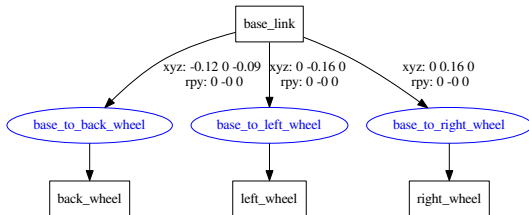
<param name="robot_description" command="$(find xacro)/xacro.py
  $(find myrobot)/urdf/myrobot.xacro" />
```

URDF Visualization

- ▶ rviz
- ▶ urdf_to_graphviz from the liburdfdom-tools (sensor model not included)

Example

```
$ urdf_to_graphviz catkin_ws/src/myrobot/urdf/simple.urdf
```





TF – Transform

What is tf

is distributed coordinate frame tracking system.

Type of nodes

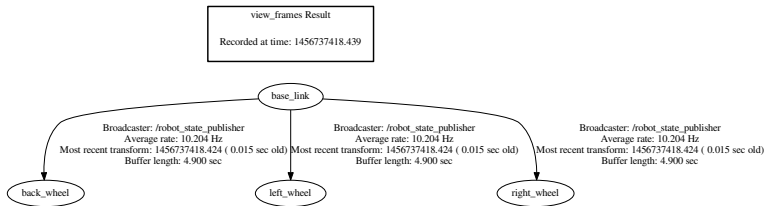
Listeners: Cache all data up to the cache limit

Publisher: Publish transforms between coordinate frames

Note: No central source for tf

TF – Data Model and Transforms

```
roslaunch tf view_frames
```



Looking for a transform:

```
roslaunch tf tf_echo base_link right_wheel
```

At time 1456738990.924

- Translation: [0.000, 0.160, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
in RPY (radian) [0.000, -0.000, 0.000]
in RPY (degree) [0.000, -0.000, 0.000]



TF – Publish a transformation

Static transform via a launch file

```
<!-- static_transform_publisher x y z yaw pitch roll frame_id child_frame_id period_in_ms -->  
<!-- static_transform_publisher x y z qx qy qz qw frame_id child_frame_id period_in_ms -->  
  
<node pkg="tf" type="static_transform_publisher" name="camera_link_broadcaster"  
      args="0.0 0.0 0.0 1.57 0 0 /world /camera_link 100" />
```

Example adding a transform using python

```
#!/usr/bin/env python  
import rospy, roslib, tf  
  
if __name__ == '__main__':  
    rospy.init_node('tf_broadcaster')  
    br = tf.TransformBroadcaster()  
    rate = rospy.Rate(10.0)  
    while not rospy.is_shutdown():  
        br.sendTransform((0.0, 2.0, 0.0), (0.0, 0.0, 0.0, 1.0),  
                        rospy.Time.now(), "world", "camera")  
        rate.sleep()
```


URDF & TF Limitations

- Topology: tree structure - it does not allow a closed loop structure.

Exercise 1

- ▶ Look at the package structure
- ▶ Open package.xml in myrobot

```
# Terminal  
ros_local  
roscd myrobot  
ls
```



Launch Simple Robot

Simple Robot

```
# Terminal 1
ros_local
roslaunch myrobot simple.launch
```

```
# Terminal 2
ros_local
rostopic list
rviz
```

```
# Terminal 3
ros_local
cd workshop/ros_robot
roslaunch tf view_frames
```

- ▶ Try different joint values
- ▶ Look at the robot model

Exercise 2

```
# Terminal 1
ros_local
roslaunch myrobot full_robot.launch
```

```
# Open workshop/ros_robot/src\
# /fake_odom/src/fake_odom.py
ros_local
roslaunch fake_odom fake_odom.py
```

- ▶ install ROS Sensor Message on your mobile phone
- ▶ open ROS Sensor Messages and set master URI to your IP
- ▶ look at the new topic (rostopic echo /TOPICNAME)
- ▶ Write a ROS Python script that transforms the sensor messages into geometry_msgs/Twist and publishes them to the topic /fake_nav/drive
- ▶ Can you steer your simple robot?

```
# Tip:
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Imu
# Use:
move = Twist(); move.angular.x=0; move.linear.x=0
```



Possible Solution

```
#!/usr/bin/python

import rospy, roslib, sys
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Imu

def callback(imu_data):
    move = Twist()
    move.angular.x=0
    move.linear.x=0
    if ( imu_data.linear_acceleration.y < -0.5 ):
        move.linear.x = imu_data.linear_acceleration.y * -1 / 10.0 / 120.0
    if (imu_data.linear_acceleration.x < -0.5) or (imu_data.linear_acceleration.x > 0.5):
        move.angular.x = imu_data.linear_acceleration.x / 10
    twist_pub.publish(move)

if __name__ == '__main__':
    '''Initializes and cleanup ros node'''
    rospy.init_node('mobile_drive', anonymous=True)
    twist_pub = rospy.Publisher("/fake_nav/drive",Twist)
    imu_subsc = rospy.Subscriber("/imu",Imu, callback, queue_size = 1)
    try:
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down ROS module"
```



ROS

Environment

URDF & TF

RoboCup Junior

Odometry and Obstacle Avoidance

OpenCV

Simulators

Further Topics



RoboCup Junior

Different Challenges

different Age Groups: **11-14** and **15-18**

Teams up to 5

Event – OnStage, Rescue, Soccer

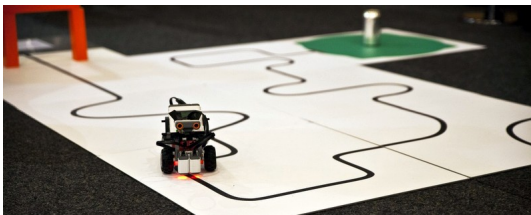
(<http://robocupjunior.at>)



RoboCup Junior

Rules

newest Rules (<http://rcj.robocup.org/>)





ROS

Environment

URDF & TF

RoboCup Junior

Odometry and Obstacle Avoidance

OpenCV

Simulators

Further Topics

Odometry

use of data from sensors to estimate robots position relative to a starting location.

Example: two wheeled differentially steered robot

The location is constantly updated – heading and distance are calculated first:

$$distance = \frac{(left_encoder + right_encoder)}{2.0} \quad (1)$$

$$theta = \frac{(left_encoder - right_encoder)}{WHEEL_BASE} \quad (2)$$

where WHEEL_BASE is the distance between the two differential drive wheels.

Odometry: Calculate position

With trigonometry we can calculate the robot's position in 2D Cartesian space:

$$\theta_{deg} = \theta * (180/\pi) \quad (3)$$

$$X_{position} = distance * \sin(\theta_{deg}) \quad (4)$$

$$Y_{position} = distance * \cos(\theta_{deg}) \quad (5)$$

Simple Motor Control

```
Motor1_Pin02 = 16; Motor1_Pin07 = 18; Motor1_Pin01 = 22
Motor2_Pin15 = 23; Motor2_Pin10 = 21; Motor2_Pin09 = 19
```

```
for iopin in (Motor1_Pin02, Motor1_Pin07, Motor1_Pin01,
             Motor2_Pin15, Motor2_Pin10, Motor2_Pin09):
    GPIO.setup(iopin,GPIO.OUT)
```

```
print "Going forwards"
GPIO.output(Motor1_Pin02,GPIO.HIGH)
GPIO.output(Motor1_Pin07,GPIO.LOW)
GPIO.output(Motor1_Pin01,GPIO.HIGH)
```

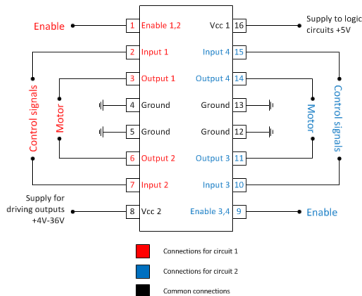
```
GPIO.output(Motor2_Pin15,GPIO.HIGH)
GPIO.output(Motor2_Pin10,GPIO.LOW)
GPIO.output(Motor2_Pin09,GPIO.HIGH)
```

```
print "Going backwards"
GPIO.output(Motor1_Pin02,GPIO.LOW)
GPIO.output(Motor1_Pin07,GPIO.HIGH)
GPIO.output(Motor1_Pin01,GPIO.HIGH)
```

```
GPIO.output(Motor2_Pin15,GPIO.LOW)
GPIO.output(Motor2_Pin10,GPIO.HIGH)
GPIO.output(Motor2_Pin09,GPIO.HIGH)
```

```
print "stop"
GPIO.output(Motor1_Pin01,GPIO.LOW)
GPIO.output(Motor2_Pin09,GPIO.LOW)
```

```
GPIO.cleanup()
```



2 x DC Motor (a 3-5 Euro)
1 x L293D (H-Bridge, 1-2 Euro)

Odometry Problems

Example: slip between wheels (1cm)

$$\begin{aligned} headingerror &= (left - right) / WHEEL_BASE \\ &= (0.01 / 0.1) \\ &= 0.1 * (180 / \pi) = 5.72 \text{ Degrees} \end{aligned}$$

Final position of robot after 10 meters:

$$X' = 10 * \sin(5.72) = 0.99 \text{ meter}$$

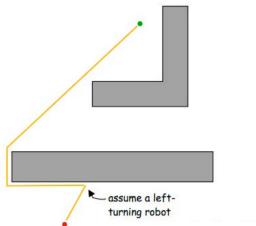
$$Y' = 10 * \cos(5.72) = 9.9 \text{ meter}$$

More or less shift of 1 meter in x direction! Small errors in theta result in large errors in X and Y.

Obstacle Avoidance

Bug 0 algorithm

- ▶ Head towards goal
- ▶ Follow obstacles until you can head to the goal again
- ▶ Repeat until successful





Obstacle Avoidance

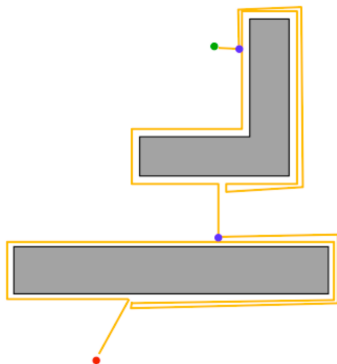
Bug 0 algorithm



Obstacle Avoidance

Bug 1 algorithm

- ▶ When encountering an obstacle, circumnavigate and map it
- ▶ Then head to goal from point of closest approach



Obstacle Detection

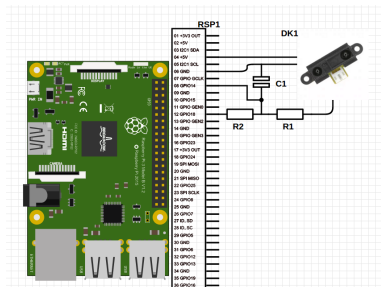
Simple Range Sensor

Common example: SHARP 2Y0A21 (10 Euro)

Analog Signal, Distances: 5cm to 80cm

Howto convert an analog to digital Signal

```
for i in range(300000):
    if GPIO.input(14):
        GPIO.output(18, 0)
        u1=u1+1
    else:
        GPIO.output(18, 1)
        u2=u2+1
u=u1-u2
u=u*2100 # 3.3V calibration
u=u/60000
u=u-1500 # 0V calibrating
```



$R1=800k-1M$, $R2=150-300K$, $C1=10-20nF$



ROS

Environment

URDF & TF

RoboCup Junior

Odometry and Obstacle Avoidance

OpenCV

Simulators

Further Topics

What is OpenCV

- ▶ Open Source Computer Vision Library (opencv.org)
- ▶ Original developed by Intel
- ▶ More than 2500 optimized Algorithms
- ▶ C/C++/Python/Java Api
- ▶ Cross-Platform
- ▶ Released under BSD (most of the algorithms)
- ▶ Current release: 2.4.4 | 3.1

Applications

- ▶ Human Computer Interfaces
- ▶ Image Processing
- ▶ Object Identification and Recognition (e.g. Face, Gestures)
- ▶ Motion Tracking
- ▶ Mobile Robotics

OpenCV Structure

- ▶ `core`: Basic structure and algorithms
- ▶ `imgproc`: Image Processing algorithms (image filtering, geometrical image transformations, histograms,...)
- ▶ `video`: video analysis (motion estimation, object tracking, ...)
- ▶ `highgui`: built-in simple UI
- ▶ `calib3d`: camera calibration and 3D reconstruction
- ▶ `features2d`: feature detectors and descriptors, feature matching
- ▶ `objdetect`: object detection (faces, eyes, people, ...)
- ▶ `ml`: machine learning for statistical classification, regression and clustering of data
- ▶ `gpu`: GPU-accelerated algorithms

Image structure

Mat:

- ▶ primary image structure in OpenCV 2.x
- ▶ (more or less) automatic memory management
- ▶ two data parts: matrix header (information about the matrix), pointer to the matrix containing pixel values.

Image Processing with Raspicam and ROS

Example on Raspi

Terminal 1

```
ros_local
```

```
roslaunch raspicam test.launch
```

Have a look at workshop/scripts/ros_openCV.py

```
ros_local
```

```
python workshop/scripts/ros_openCV.py
```

Exercise 2

- ▶ Play around with different Feature detectors
- ▶ copy the script and modify it that it detects lines



Image Processing with Raspicam and ROS

Possible Solution

```
# exchange lines 59 - 76 (see py_houghlines documentation)
# convert to gray image and use Canny Edge detector
gray = cv2.cvtColor(image_np,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize = 3)

# min Line length and Gap between lines
minLineLength = 100
maxLineGap = 10

lines = cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,maxLineGap)
for x1,y1,x2,y2 in lines[0]:
    cv2.line(image_np,(x1,y1),(x2,y2),(0,255,0),2)
```



ROS

Environment

URDF & TF

RoboCup Junior

Odometry and Obstacle Avoidance

OpenCV

Simulators

Further Topics

Simulators

- ▶ STDR 2D <http://stdr-simulator-ros-pkg.github.io>
- ▶ MORSE <https://www.openrobots.org/morse/doc/stable/morse.html>
- ▶ V-REP <http://www.v-rep.eu>
- ▶ Gazebo <http://gazebo-sim.org>
- ▶ Baxter (Gazebo) http://sdk.rethinkrobotics.com/wiki/Baxter_Simulator
- ▶ AR Drone Simulator http://wiki.ros.org/tum_simulator
- ▶ ...

Gazebo Features

Features



Dynamics Simulation

Access multiple high-performance physics engines including [ODE](#), [Bullet](#), [Simbody](#), and [DART](#).



Advanced 3D Graphics

Utilizing [OGRE](#), Gazebo provides realistic rendering of environments including high-quality lighting, shadows, and textures.



Sensors and Noise

Generate sensor data, optionally with noise, from laser range finders, 2D/3D cameras, Kinect style sensors, contact sensors, force-torque, and more.



Plugins

Develop custom plugins for robot, sensor, and environmental control. Plugins provide direct access to Gazebo's [API](#).



Robot Models

Many robots are provided including PR2, Pioneer2 DX, iRobot Create, and TurtleBot. Or build your own using [SDF](#).



TCP/IP Transport

Run simulation on remote servers, and interface to Gazebo through socket-based message passing using Google [Protobufs](#).



Cloud Simulation

Use [CloudSim](#) to run Gazebo on Amazon, Softlayer, or your own OpenStack instance.



Command Line Tools

Extensive command line tools facilitate simulation introspection and control.

Figure : Gazebo features taken from <http://gazebo.sim.org/>

Extending your URDF/XACRO for Gazebo

<collision>

Important for determining how it will interact with other objects.

<inertial>

Defines the mass and moment of inertia of the link.

Needed to move it according to Newton's laws.

<material>

Material of visual element, must also be specified for each link.

Efficient Robot Models

Physics (CPU)

Limit contacts (`<physics max_contacts="3"/>`)

Kinematic trees are better than loops

Reduce number of joints in a model

Collisions (CPU)

Primitives are more efficient than trimeshes

Limit collision mesh size (`< 5k triangles per link`)

Rendering (GPU)

Limit visual mesh size (`< 5k triangles per link`)

Limit image/depth sensor resolution or rate

Working with inertial values

View → Center of Mass/Inertia: visual representation of the inertia matrix and mass for each link.

If the inertial data is very different from (much smaller or larger) the visual or collision geometry, then you have a problem.

Gazebo Differential Drive Controller

```
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <leftJoint>base_to_left_wheel</leftJoint>
    <rightJoint>base_to_right_wheel</rightJoint>
    <robotBaseFrame>base_link</robotBaseFrame>
    <wheelSeparation>0.3</wheelSeparation>
    <wheelDiameter>0.12</wheelDiameter>
    <publishWheelJointState>true</publishWheelJointState>
  </plugin>
</gazebo>
```

Starting Gazebo with adapted model

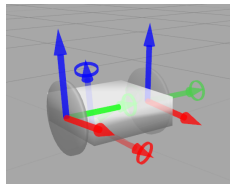
```
<launch>
  <!-- Load the URDF model into the parameter server -->
  <param name="robot_description" textfile="$(find myrobot)/urdf/simple-gazebo.urdf" />

  <!-- Start Gazebo with an empty world -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch"/>

  <!-- Spawn a our robot in Gazebo, using the description from the parameter server -->
  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
    args="-param robot_description -urdf -model simple-gazebo" />
</launch>
```

Using teleop-twist-keyboard:

```
roslaunch teleop_twist_keyboard \
  teleop_twist_keyboard.py
```





ROS

Environment

URDF & TF

RoboCup Junior

Odometry and Obstacle Avoidance

OpenCV

Simulators

Further Topics



Further Tips and Tricks

`ros_control`

Use this package to interface your Hardware.

`ros_serial`

Use `ros_serial` for windows, embedded, maybe even rt systems.

`rosh`

Python-based scripting and runtime environment for ROS.

use standard topics and msgs

odom: `nav_msgs/Odometry`

cmd_vel: `geometry_msgs/Twist`

scan: `sensor_msgs/LaserScan`

diagnostics: `diagnostic_msgs/DiagnosticStatus`

Further Topics

- ▶ Virtualisation (virtualbox) and Container (docker)
- ▶ Autonomous navigation in unknown and known environments
- ▶ Self localisation and homography
- ▶ ...

Further Information

Slides and Code can be downloaded from:

```
git clone https://git.uibk.ac.at:c703101/prototyping-robots.git
```

Raspberry Image with ROS Indigo can be downloaded from:

```
https://iis.uibk.ac.at/public/simon/raspberry_ros.img.tgz
```

Installation from image:

```
tar xzf raspberry_ros.img.tgz
```

```
sudo dd if raspberry_ros.img of=/dev/mmcblkX 2
```

² where /dev/mmcblkX is the sd card



Further Information

Programming Robots with ROS, by Morgan Quigley, Brian Gerkey and William D. Smart, O'Reilly, 2016.

ROS By Example Volume 2, by R. Patrick Goebel, Lulu Enterprises, 2015.

Learning Robotics Using Python, by Lentin Joseph, Packt Publishing, 2015.

MORSE Simulator <https://www.openrobots.org/morse>.

Tutorials: <http://wiki.ros.org/>.

RosCon: Video Tutorials on youtube and vimeo.

Docker: <http://docs.docker.com>.