Introducing rosc

ROSCon 2013

Stuttgart, 12th May 2013
Steps towards Embedded ROS

Common architecture supported by ROS today
Steps towards Embedded ROS

Common architecture supported by ROS today

Established, but has some drawbacks:

- Control loops being closed over fieldbuses
  - $1 \text{ kHz} / 10 \text{ axes}$ is mostly the limit
  - Safety-criticality
- Motion Control fights against all the rest on the PC
  - Bad motion control quality
  - Less resources for higher intelligence
Steps towards Embedded ROS

Architecture requested (and delivered by rosc & µROSnode)
Steps towards Embedded ROS

Architecture requested (and delivered by rosc & μROSnode)
Instantaneous superposition of platform translation and rotation

- **moving straight**: wheels rotate in same direction
- **moving sideways**: wheels rotate in opposite directions
- **moving diagonally**: two diagonally opposite wheels stand still, two others rotate in same direction
- **rotate around central axis**: wheels on one side rotate in opposite direction to wheels on other side

Meccanum wheel with passive rollers

Pictures courtesy of KUKA Laboratories
Industry Default: Centralized Motion Control Architecture

Centralized Dynamics Model & Control using a PC

- 500 rpm!
- 350 rpm!
- 800 rpm!
- 1200 rpm!

Pictures courtesy of KUKA Laboratories
Advanced Alternative: Distributed Motion Control Architecture

Dynamics Model & Control running on Motor Control Level

Abstract translation commands: „Move to X,Y + a° + mm/s !“

Same story for manipulators

Pictures courtesy of KUKA Laboratories
Steps towards Embedded ROS

„Embedded“
- ARM Cortex A (32bit): Beagle, Panda, Rasp, ODROID, Phytec
- Intel Atom (64bit): Congatec, Kontron

„Small Embedded“
- ARM Cortex M (32bit): mbed
- AVR (8-32bit): Arduino
- XMOS (32bit): Synapticon SOMANET

Perception & Intelligence
Control & Data Acquisition
Steps towards Embedded ROS

**rosc**

- ROS client library for Small Embedded devices
  - XML-RPC
  - TCPROS
- Light-weight (< 32 kB memory on XMOS*)
- No dependencies
- ANSI C (99)
Steps towards Embedded ROS

- ROS client library for Small Embedded devices
- XML-RPC
- TCPROS
- Lightweight (< 32 kB memory on XMOS*)
- No dependencies
- ANSI C (99)

* XMOS Arch:
  - 32bit
  - Multicore (4-32)
  - 125 Mhz/Core
  - 64kB RAM/Tile
  - Hard real-time
<table>
<thead>
<tr>
<th>Aspect</th>
<th>µROSNode</th>
<th>rosc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>University project</td>
<td>Commercial open-source</td>
</tr>
<tr>
<td>Status</td>
<td>Working demos available</td>
<td>70% of first release</td>
</tr>
<tr>
<td>Long-term</td>
<td>No certain plans</td>
<td>Aims to be major ROS Industrial/Products client lib</td>
</tr>
<tr>
<td>Memory</td>
<td>Compact (1 MB reference device)</td>
<td>As small as possible (64 kB reference device)</td>
</tr>
<tr>
<td>OS</td>
<td>ChibiOS &amp; POSIX reference</td>
<td>Bare-metal reference</td>
</tr>
<tr>
<td>Transport</td>
<td>OS driver (LWIP)</td>
<td>Modular transport layer concept</td>
</tr>
</tbody>
</table>
Components of rosc

- Your (ROS compatible) code
- rosc main
- rosc OS support pkg
- rosc bare-metal comm pkg
- rosc Arch support pkg
Building a node with rosc

Your (ROS compatible) code

rosc main

rosc OS support pkg

rosc bare-metal comm pkg

rosc Arch support pkg

Executable

Flashable binary

Executable

Flashable binary
The XML-RPC pain

Topic Request

<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
    <methodName>topic_request</methodName>
    <params>
        <param>
            <value>
                <array>
                    <data>
                        <i4>1</i4>
                    </data>
                </array>
            </value>
        </param>
        <param>
            <value>
                <array>
                    <data>
                        <value>TCPROS</value>
                        <value>ROS</value>
                        <i4>41776</i4>
                    </data>
                </array>
            </value>
        </param>
    </params>
</methodCall>

<methodResponse>
    </params>
</methodResponse>
The XML-RPC pain

What common developers think about it

What embedded developers think about it
Current features of rosc (1)

HTTP/XML Parser

- Size: < 10 kB (on XMOS)
- Time for parsing: 139µs
  (publisherUpdate msg on XMOS, 125MHz, 100 byte buffer)
- Features/Limitations:
  - Streaming Parser
    (parsing on demand, fully variable buffersize 1 Byte - XXX Bytes)
  - Almost fully validating
    (unknown tags can not be validated)
  - Does not support any encoding
    (e.g. gzip, due to it's streaming nature)
  - Maximum depth for tags: 20
    (can be set to any value by #define)
Current features of rosc (2)

XML-RPC Message Generator
• Size: < 7 kB (on XMOS)

Port Interface Handling (Services, Topics, XML-RPC)
• Limitations on bare-metal systems:
  Fixed amount of ports and thus a limited amount of possible connections

ROS msg Header/Source Generation

TCPROS (Un-)Marshalling

+ Future dev towards ROS Industrial/Products
Steps towards Industrial/Product ROS

Solution requested (and delivered by rosc & μROSnode)
Steps towards Industrial/Product ROS

ROS on all levels (not possible using today’s ROS!)
Requirements for a future-proof (Embedded) ROS

- Transport-independent (TCP/IP not required)
- Standard application protocols & reference architectures (Compatibility)
- Real-time capable (msg queues, transport)
- Replacement of XML-RPC (by JSON, or even ROSRPC?…)
- No master anymore (at least multi-master support)
- Model-based toolchain support
- Long-term support by foundation & suppliers
- Quality metrics & automated QA process
Requirements for a future-proof (Embedded) ROS

• Transport-independent (TCP/IP not required)
• Standard application protocols & reference architecture (Compatibility)
• Real-time capable (msg queues, transport)
• Replacement of XML-RPC (by JSON, or even ROSRPC?...)
• No master anymore (at least multi-master support)
• Model-based toolchain support
• Long-term support by foundation & suppliers
• Quality metrics & automated QA process
Mainly ROS is not RT-capable because of:

- IP transport
- Client implementations not supporting RTOS mechanisms
Draft for a RT-capable, master-free rosc

RT-capable ROS MW needs to support RT
- Multi-threading
- Scheduling
- Transport