Embedded ROS

By Paul Bouchier

Do you design robots? You probably ponder how you should partition functions into subsystems embedded in the robot. If you use ROS in your robot, you have additional concerns around how to integrate ROS's high-level functions with lower-level subsystems. You want to understand current and future design alternatives advantages and tradeoffs.

Embedded Systems in Robots

There are higher-level subsystems embedded in a robot, which are responsible for domains such as vision, reasoning, and planning. Today, software dependencies necessitate running these ROS applications on full-blown Ubuntu Linux PCs—high-cost computers with an appetite for power. This is fine for low-volume robots that are not cost sensitive. But the need for high-cost PCs to run ROS is changing to the benefit of robot designers.

There is current development work focused on providing binary installs for ARM processor-based platforms, including the popular Raspberry Pi board. These platforms will dramatically reduce cost, power consumption, and the physical size of each ROS server. Clustering low-cost ARM servers will allow distributing ROS subsystems to achieve scale-out expansion. Reduced software dependencies will mean more configuration options for the big brains of the robot, ranging from Ubuntu to stripped-down configurations.

Complementing the higher-level subsystems are small embedded systems dedicated to low-level control and connecting devices (e.g., sensors, actuators, and so on) whose electrical interfaces are not available from a server running ROS. These small embedded systems must be cheap to enable manufacturing more robots at a lower cost. Embedded system costs can be as low as US$1.50 for a small system, to US$10 and up for a larger system. These embedded systems are developed with a focus on keeping parts costs low and performance high for a very limited set of tasks. These low-level systems are too small to run ROS applications, but must be able to communicate with them.

Real Time

Robot motion occurs in real time. Robot designers care how that motion occurs. Motion must occur with predictable timing and must meet timing deadlines defined by the application.

ROS runs on Linux, which does not provide timing guarantees. The need for timing guarantees drives robot designers to partition robots into real-time and nonreal-time subsystems. It has the additional benefit of narrowing the focus of safety and other critical reviews to simpler subsystems.

Attaching embedded real-time systems to ROS is one way of reaping the benefits of ROS' higher-level capabilities while meeting real-time system needs. A current example of this technique is found in ROS *Industrial*. ROS on Linux plans the motion of a robot arm and passes the plan to a controller that executes it, moving the arm in real-time.

A second possible approach to real-time needs that I suggest, though not yet seen in research, is to port some ROS packages to a version of Unix that will offer real-time guarantees. Designers would need to review the design of those packages from a real-time perspective. It remains to be seen whether the ARM processor support work will enable ROS on a real-time Unix.

Approaches for Using ROS with Embedded Systems

ROS planners chose wisely to focus on enabling the higher levels of robot intelligence. They obtain greater value by enabling researchers to collaborate on different parts of the higher-level software stack; ROS would not have become

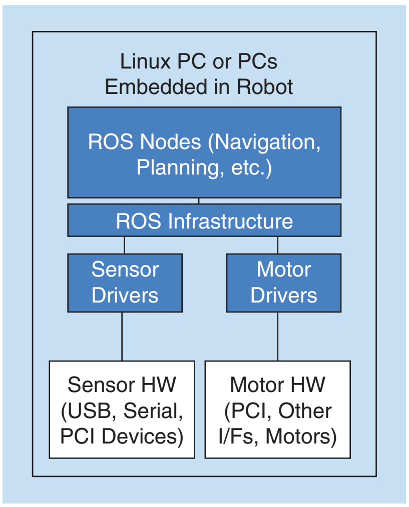

Linux PC or PCs Embedded in Robot
ROS Nodes (Navigation, Planning, etc.)
ROS Infrastructure
Sensor Drivers
Motor Drivers
Sensor HW (USB, Serial, PCI Devices)
Motor HW (PCI, Other I/Fs, Motors)

Figure 1. The embedded ROS PC controls peripherals.

Digital Object Identifier 10.1109/MRA.2013.2255491
Date of publication: 6 June 2013

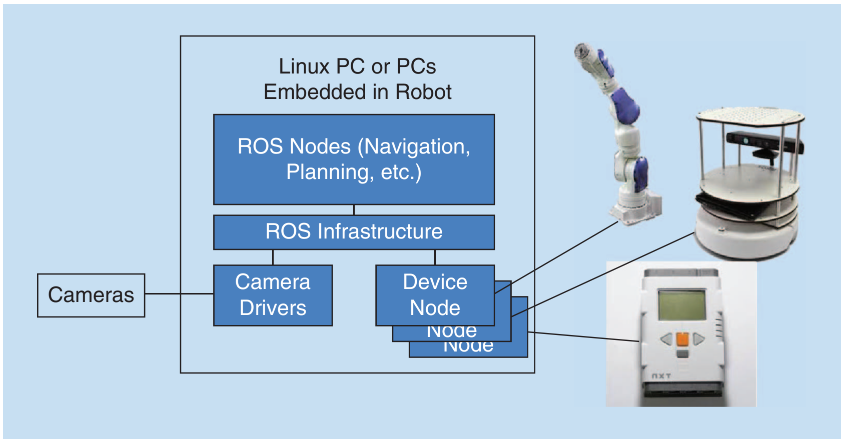


Figure 2. Custom nodes control devices and a robot.

as successful had it focused on motors and sensors. However, connection to embedded systems is lightly addressed, with the robot designer having to marry different kinds of subsystems. Fieldbuses are not supported.

There are three architecture styles the robot designer can use to embed ROS into robots.

Embedded ROS PC

An industrial PC is fitted with motor control and other cards, and runs ROS/Linux (Figure 1).

This architecture offers smooth integration with ROS and the ability to run other nodes on the embedded PCs. It is also complicated to configure: Linux real-time extensions should be installed; but even so, Linux is not a real-time OS (RTOS). Frame rates may be limited and jitter can be excessive. Smart motor controllers can help. Memory should be locked down to prevent swapping, but other traps await the unwary.

Proprietary Embedded System with Custom Interface

A variety of special-purpose embedded systems, some with an RTOS, can provide a range of control options to the robot designer. An entire proprietary robot can be managed from ROS using this kind of control interface, as shown in Figure 2.

An ROS device node on one of the ROS systems translates between ROS nodes and a proprietary interface to the embedded subsystem, publishing or consuming messages. It has the advantage that the translation node abstracts low-level details from higher-level ROS applications. In addition, the embedded subsystem can be designed to provide real-time guarantees. A recent development in this area is ros_arduino_bridge, which enables the device node to get and set pin data on an Arduino.

ROS Messaging and APIs Extended to Embedded Systems

The interprocess communication architecture of ROS is centered on remote procedure calls (RPCs) with publish/subscribe support. These are used to exchange standard or custom messages between ROS nodes. The robot designer can use two different approaches to pass these messages to embedded systems.

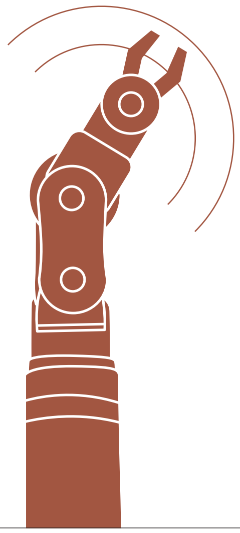
Rosserial is an approach, shown on the right of Figure 3. It provides a proxy that relays messages over a link to a C++ client on the embedded system. The rosserial client on the embedded system does not depend on an OS, and provides an ROS-like API to embedded system software, enabling it to publish, subscribe, and offer and consume RPC services. Rosserial is easily ported to any platform that supports C++. Ports currently exist for Arduino, embedded Linux, and Xbee, with wireless and wired link support. Multiple embedded systems are supported by using multiple proxy instances.

Reflexxes Motion Libraries

Reflexxes

Robust, reliable, broadly tested, and utilized by renowned robot manufacturers.

Reflexes for Robots



Instantaneous & deterministic reactions to unforeseen sensor signals

Smooth motion generation within one millisecond

Multi-platform support (QNX, VxWorks, Linux, Windows, Mac OS X, and more)

Free for academic use

Free downloads and evaluation packages

www.reflexxes.com

A second approach, shown on the left of Figure 3, is enabled by the recent (alpha) release of the uros package, and the anticipated release of the rosc package. Both are written in C and built for direct connection to Ethernet; they handle native ROS connections and messages.

Rosbridge offers a third alternative: a proxy provides dynamic socket and web-socket-based access to the full capabilities of ROS. This allows an embedded ROS application to interact with software in the cloud.

The general approach of sending ROS messages to the embedded system brings several important benefits to robot designers. The embedded system can be designed to provide real-time guarantees for its software, and may even run an RTOS. Seamless transport of ROS messages between higher and lower levels, and a consistent ROS API, makes design easier. Unified logging eases debugging.

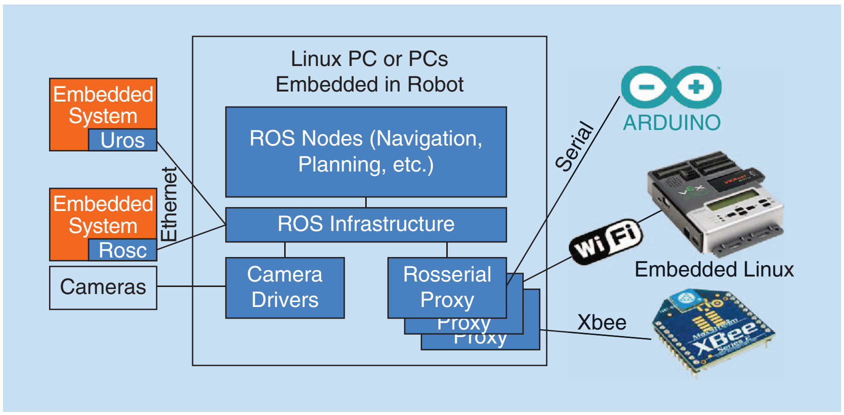


Figure 3. ROS messages passed to embedded system.

Rosbag can capture and play-back messages to and from the embedded systems for better analysis. Tradeoffs related to rosserial are that the proxy could be a bottleneck, and the rosserial client is written in C++. Concerns related to uros and rosc are that a small embedded system may be overwhelmed by the overhead of TCP/IP and XMLRPC processing.

As you design robots, think about the advantages and tradeoffs for the architectural alternatives presented above. Although the general approaches are appropriate for many higher-level frameworks, ROS offers explicit support that enables them. Choose an architecture with the right qualities and make sure the tradeoffs do not hurt your design.

Robotnik
Advanced Robotic Solutions
KEY FEATURES:
>> ROS architecture
>> High mobility and modularity
>> Indoor / Outdoor Navigation
>> Suitable for research, security and inspection
>> Customized Solutions
Contact us:
info@robotnik.com
www.robotnik.eu
Robotnik