

Low-Cost Manipulation Powered by ROS

Morgan Quigley, Alan Asbeck, and Andrew Y. Ng

Computer Science Department, Stanford University

{mquigley, aasbeck, ang}@cs.stanford.edu

Introduction

In recent years, substantial progress has been made towards real-world usage of robots for home and office automation. Continued advances in computational power, sensing, and battery technology have resulted in rapid progress towards true “personal robots.” However, the cost of state-of-the-art platforms remains a daunting challenge to practical deployments, as do human-robot interaction and safety concerns. Furthermore, as robots continue to become more sophisticated, software integration, performance, and reliability issues are becoming ever more important. Our AAAI 2010 Robotics Workshop exhibit, demonstrated our recent work in both low-cost manipulation and large-scale robotics software integration. The following sections discuss specific aspects of mechanism and software design.

Low-cost Manipulation

We are investigating several techniques for reducing the cost of robotic manipulation. Our goal is a manipulator with a roughly anthropomorphic workspace, 2-kilogram payload, several-millimeter repeatability, and a \$3000 (or lower) bill of materials. We have investigated several cost-reduction approaches, and our current work aims to exploit the mechanical simplifications and reduced control-system requirements made possible by introducing compliant elements such as springs and elastomers in the manipulator drivetrain. Our approach also aims to employ grounded or nearly-grounded actuators (i.e., motors mounted in the torso or shoulder of the robot, as opposed to flying on the manipulator), resulting in a lightweight manipulator which can be readily fabricated using rapid-prototyping methods in wood or sheet metal.

The manipulator we brought to the AAAI 2010 Robotics Workshop is photographed in Figure 1. This 7-dof manipulator was prototyped using laser-cut plywood and off-the-shelf mechanical hardware. The first four degrees of freedom are driven by commodity stepper motors: one pair in the torso and one pair in the first link of the shoulder. These four degrees of freedom have series-elastic couplings between remote cable-driven capstans and their respective links, which decouples the motor inertia from the link. As has been well documented in the robotics literature, series-

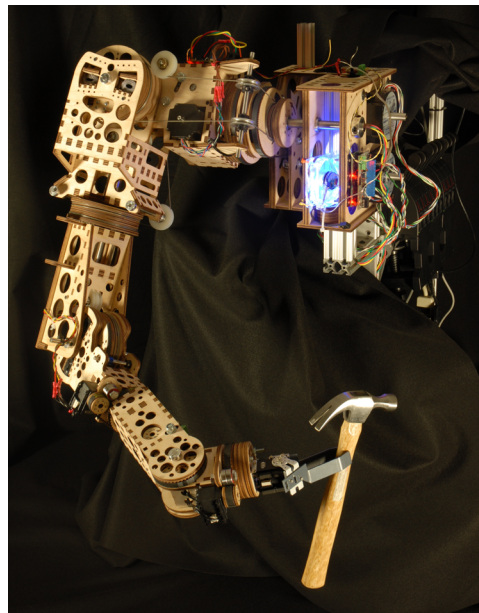


Figure 1: The low-cost compliant manipulator shown at the AAAI 2010 Robotics Workshop

elastic designs sacrifice control bandwidth, but have improved human safety, simpler interactions with the environment, and intrinsic force sensing. We have found experimentally that these properties of series-elastic actuation are a good match for our low-cost design goals: relaxing the mechanical tolerances of manipulator manufacture does not dramatically reduce bandwidth beyond the inherently low control bandwidth of series-elastic designs.

To demonstrate the dexterity of the manipulator in a live setting, we constructed a teleoperation scheme by affixing four magnetometers and inertial sensors to a shirt, such that they were tightly attached to the teleoperator’s torso, upper right arm, lower right arm, and right hand. This allowed recovery of the teleoperator’s hand position and orientation, which were fed into an inverse-kinematics solver to compute joint-angle targets for the manipulator. At the AAAI 2010 Robotics Workshop, we used this control scheme to play a game of chess, as shown in Figure 2.

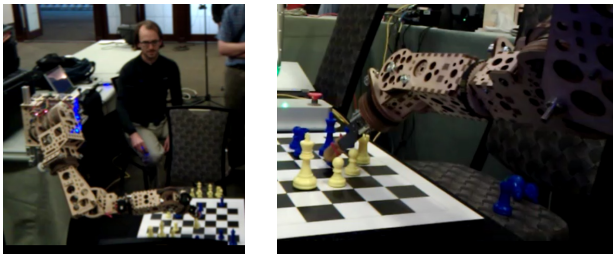


Figure 2: Playing chess via teleoperation: arm motions of a human teleoperator were transformed into joint-angle targets for the low-cost manipulator.

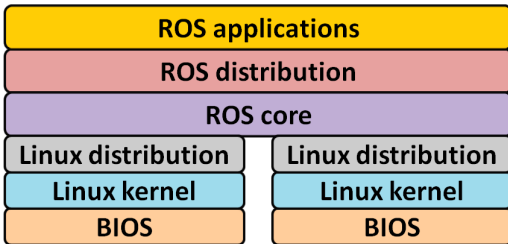


Figure 3: A high-level view of the ROS software stack, illustrating that ROS layers on top of traditional operating systems and can facilitate communications between multiple machines.

Robotic Middleware: ROS

In collaboration with researchers at Willow Garage and elsewhere, we have developed the Robotic Operating System (ROS), a framework for large-scale integration of robotic software. Although ROS itself is one of dozens (if not hundreds) of message-passing frameworks available, we believe that its low-level architecture of anonymous publish/subscribe and “non-intrusive” implementation pattern increases its usability and relevance for robotics research, and (importantly) allows easier code re-use from other robotics frameworks or standalone libraries.

Specifically, the design of ROS encourages low-level software to make few assumptions about higher layers of the software stack, and the ROS client library itself makes very few assumptions about the program structure hosting it. This philosophy can be summarized as “we don’t wrap `main()`”, and allows a variety of software-engineering paradigms to be employed, such as component-based graphical structures (e.g., an image-processing pipeline), object-oriented remote procedure calls (e.g., a knowledge base), or, most commonly, hybrid systems which use a combination of message-based and transaction-based structures.

Furthermore, we designed ROS to support hot-swapping of modules in a running system, a capability which greatly eases development on highly complex robots such as the Willow Garage PR2, which uses dozens of ROS programs to compose the runtime environment. This hot-swap ability, coupled with the anonymous publish/subscribe architecture, is exploited by a large collection of *tools* which can interact at a meta-level with ROS message streams. For ex-

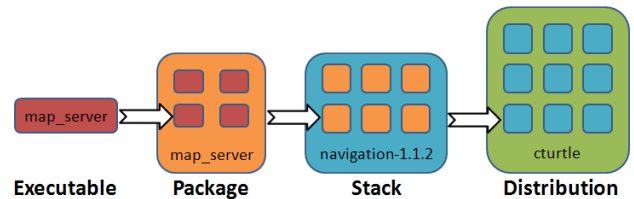


Figure 4: The ROS code organization hierarchy.

ample, logging or visualization programs can be “injected” into a running ROS system, where they subscribe to data streams without any other node being aware of their presence. Other tools perform message playback, routing, filtering, introspection, and many other operations.

The intended place of ROS within the software stack is shown in Figure 4, which emphasizes that ROS is not an operating system in the traditional sense; rather, it provides further layers of abstraction to aid the development of complex robot software.

Over the past two years, the ROS user community has grown far beyond its origins at Stanford University and Willow Garage; our software crawler indexes over 30 open-source repositories from universities and corporations around the world, which collectively host over 1200 packages (the fundamental unit of the ROS build system). These packages range from low-level device drivers to high-level executives, including components for navigation, manipulation, computer vision, human interfaces, etc.

With this enlarged user community comes the need for tools to organize and distribute large amounts of software written by many different entities. To address this challenge, we developed the organizational system shown in Figure 4. Executables and libraries are built in Packages. Collections of packages that are tested together are versioned simultaneously as Stacks. In turn, collections of stacks which are tested together are called Distributions. The intent is for ROS Distributions to serve as stable build targets, much like distributions of traditional operating systems (e.g., Windows 7, Mac OS X 10.6, Ubuntu Linux 10.04), and have a predictable 6-month release schedule.

Summary and Future Work

Our exhibit at the AAAI 2010 Robotics Workshop demonstrated our recent work in low-cost compliant manipulation as well as robotics software engineering. We intend to continue developing low-cost manipulators, exploring the use of sheet-metal fabrication techniques as well as experimenting with various non-anthropomorphic kinematic structures. The ROS-based software used in our demonstration can be downloaded from <http://stanford-ros-pkg.googlecode.com>.