ros_control: A generic and simple control framework for ROS

To cite this version:

HAL Id: hal-01662418
https://hal.archives-ouvertes.fr/hal-01662418
Submitted on 13 Dec 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
ros_control: A generic and simple control framework for ROS

Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez, Tsoyrioukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Fernandez Perdomo

1 hiDOF, Inc. (at the time of this work) 2 PAL Robotics (at the time of this work) 3 PickNik Consulting 4 Heriot-Watt University, Edinburgh Centre for Robotics 5 Department of Advanced Robotics, Istituto Italiano di Tecnologia (IIT) 6 Fraunhofer IPA 7 Clearpath Robotics 8 Honeybee Robotics 9 Kinema Systems Inc. 10 John Hopkins University 11 Willow Garage Inc. (at the time of this work) 12 Pick-it NV

Summary

In recent years the Robot Operating System (Quigley et al. 2009) (ROS) has become the ‘de facto’ standard framework for robotics software development. The ros_control framework provides the capability to implement and manage robot controllers with a focus on both real-time performance and sharing of controllers in a robot-agnostic way. The primary motivation for a separate robot-control framework is the lack of realtime-safe communication layer in ROS. Furthermore, the framework implements solutions for controller-lifecycle and hardware resource management as well as abstractions on hardware interfaces with minimal assumptions on hardware or operating system. The clear, modular design of ros_control makes it ideal for both research and industrial use and has indeed seen many such applications to date. The idea of ros_control originates from the pr2_controller_manager framework specific to the PR2 robot but ros_control is fully robot-agnostic. Controllers expose standard ROS interfaces for out-of-the box 3rd party solutions to robotics problems like manipulation path planning (MoveIt! (Chitta, Sukan, and Cousins 2012)) and autonomous navigation (the ROS navigation stack). Hence, a robot made up of a mobile base and an arm that support ros_control doesn’t need any additional code to be written, only a few controller configuration files and it is ready to navigate autonomously and do path planning for the arm. ros_control also provides several libraries to support writing custom controllers.

Figure 1: Overview
The backbone of the framework is the Hardware Abstraction Layer, which serves as a bridge to different simulated and real robots. This abstraction is provided by the `hardware_interface::RobotHW` class; specific robot implementations have to inherit from this class. Instances of this class model hardware resources provided by the robot such as electric and hydraulic actuators and low-level sensors such as encoders and force/torque sensors. It also allows for integrating heterogeneous hardware or swapping out components transparently whether it is a real or simulated robot.

There is a possibility for composing already implemented `RobotHW` instances which is ideal for constructing control systems for robots where parts come from different suppliers, each supplying their own specific `RobotHW` instance. The rest of the `hardware_interface` package defines read-only or read-write typed joint and actuator interfaces for abstracting hardware away, e.g., state, position, velocity and effort interfaces. Through these typed interfaces this abstraction enables easy introspection, increased maintainability and controllers to be hardware-agnostic.

The `controller_manager` is responsible for managing the lifecycle of controllers, and hardware resources through the interfaces and handling resource conflicts between controllers. The lifecycle of controllers is not static. It can be queried and modified at runtime through standard ROS services provided by the `controller_manager`. Such services allow to start, stop and configure controllers at runtime.

Furthermore, `ros_control` ships software libraries addressing real-time ROS communication, transmissions and joint limits. The `realtime_tools` library adds utility classes handling ROS communications in a realtime-safe way. The `transmission_interface`
package supplies classes implementing joint- and actuator-space conversions such as: simple reducer, four-bar linkage and differential transmissions. A declarative definition of transmissions is supported directly with the kinematics and dynamics description in the robot’s Universal Robot Description Format (URDF) (Willow Garage 2009) file. The \texttt{joint_limits_interface} package contains data structures for representing joint limits, methods to populate them through URDF or yaml files and methods to enforce these limits. \texttt{control_toolbox} offers components useful when writing controllers: a PID controller class, smoothers, sine-wave and noise generators.

The repository \texttt{ros_controllers} holds several ready-made controllers supporting the most common use-cases for manipulators, mobile and humanoid robots, e.g. the \texttt{joint_trajectory_controller} is heavily used with position-controlled robots to interface with MoveIt!. Finally, \texttt{control_msgs} provides ROS messages used in most controllers offered in \texttt{ros_controllers}.

\texttt{ros_control} was conceptualized by Sachin Chitta at Willow Garage Inc. and initial design and implementation was done by Sachin Chitta (then at Willow Garage), Wim Meussen, Vijay Pradeep and Eitan Marder-Epstein (then at HiDOF) before being released open-source.

\texttt{ros_control} is released as binary packages with each new version of ROS, source code is hosted at the \texttt{ros-controls} Github organization. Documentation on behaviour, interfaces, doxygen-generated pages and tutorials can be found at \texttt{ros_control} and \texttt{ros_controllers}. For a thorough presentation we invite the interested reader to watch the talk given at ROSCon2014 (Adolfo Rodríguez Tsouroukdissian, n.d.).

**Robots using \texttt{ros_control}**

Being a mature framework, \texttt{ros_control} is widely applied to both production and research platform robots. A few examples where the control system is implemented with \texttt{ros_control} are:

- The “Twil” robot at Federal University of Rio Grande do Sul (Lages 2017)
- The quadruped robots HyQ and HyQ2Max (Semini et al. 2011, Semini et al. (2017)) at Istituto Italiano di Tecnologia
- NASA’s humanoid and biped robots: Valkyrie & Robonaut (N. A. Radford et al. 2015, Hart et al. (2014), Badger et al. (2016))
- PAL Robotics’ humanoid, biped and mobile robots: REEM, REEM-C, PMB2, Tiago and Talos (Stasse et al. 2017)
- Shadow Robot’s anthropomorphic, highly sensorized and precise Shadow Hand (Meier et al. 2016)
- Universal Robots’ industrial arms: UR3, UR5 (Andersen 2015)

---

Robot names and credits in order of appearance: Valkyrie (Photo by NASA / Bill Stafford), Husky (Photo by Clearpath Robotics), OTTO (Photo by Otto Motors), TIAGo (Photo by PAL Robotics), Dexterous Hand (Photo by Shadow Robot), HyQ2Max (Photo by Istituto Italiano di Tecnologia), TALOS (Photo by PAL Robotics)

References


