Towards Autonomous Operation of Robonaut 2

Julia M. Badger Stephen W. Hart J.D. Yamokoski

The Robonaut 2 (R2) platform, as shown in Figure 1, was designed through a collaboration between NASA and General Motors to be a capable robotic assistant with the dexterity similar to a suited astronaut [1]. An R2 robot was sent to the International Space Station (ISS) in February 2011 and, in doing so, became the first humanoid robot in space. Its capabilities are presently being tested and expanded to increase its usefulness to the crew. Current work on R2 includes the addition of a mobility platform to allow the robot to complete tasks (such as cleaning, maintenance, or simple construction activities) both inside and outside of the ISS. To support these new activities, R2's software architecture is being developed to provide efficient ways of programming robust and autonomous behavior.

In particular, a multi-tiered software architecture is proposed that combines principles of low-level feedback control with higher-level planners that accomplish behavioral goals at the task level given the run-time context, user constraints, the health of the system, and so on. The proposed architecture is shown in Figure 2. At the lowest-level, the **resource level**, there exists the various sensory and motor signals available to the system. The sensory signals for a robot such as R2 include multiple channels of force/torque data, joint or Cartesian positions calculated through the robot's proprioception, and signals derived from objects observable by its cameras. The motor resources are the

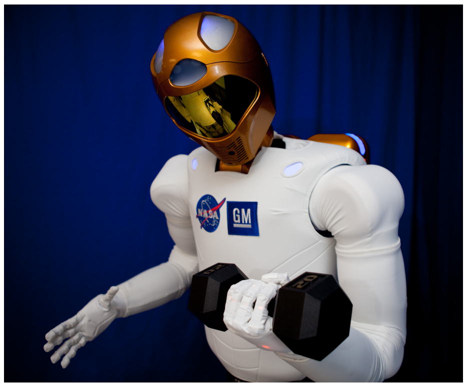

Figure 1: Robonaut 2. Credit: J. Bibby

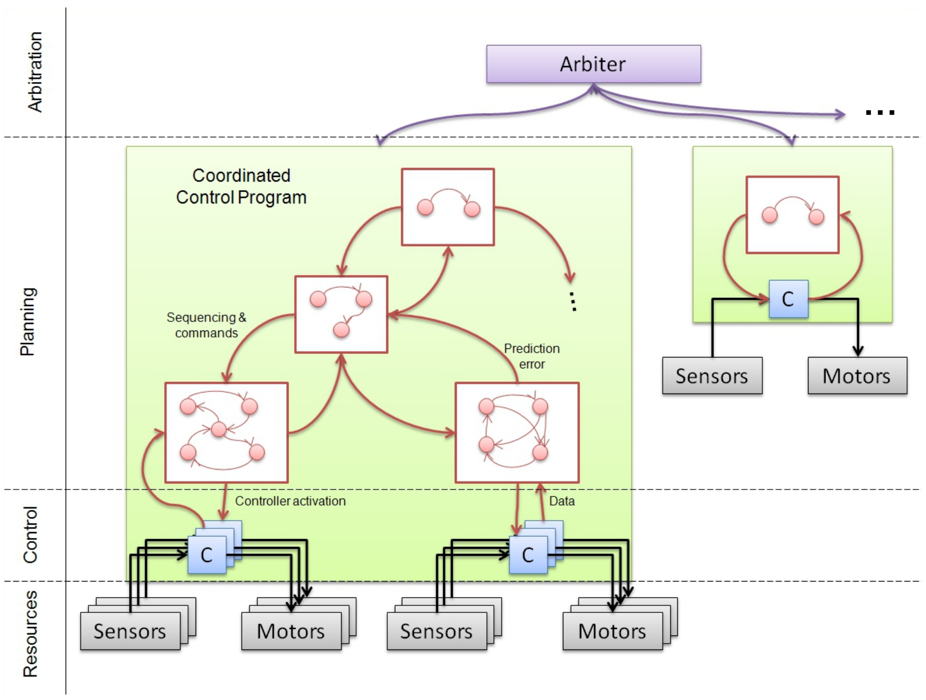

Figure 2: R2's state of the art multi-tiered control architecture.

embedded motor units that accept input commands. For R2, this includes the low-level torque control loops that run at each of R2's joints.

Sensorimotor resource signals are combined at the **control level** into multi-objective feedback controllers that optimize cost functions for tracking and/or regulating reference inputs [2, 3]. Each objective function encodes a task (e.g. desired reference position, joint limit avoidance, singularity avoidance, etc.). Tasks are assigned a priority with lower priority tasks projected into the null-space of higher priority tasks. This ensures that all objectives will be meet assuming the null-space projections do not lose rank. In cases where projections lose rank, approximations can be made to allow progress towards achieving the objective [2]. These controllers are sequenced and combined at the **planner level** into coordinated control programs that attempt to accomplish high-level goals like PickAndPlace, Grasp, PegInHole, Traverse, etc. These are the "verbs" of the system. Control at this level incorporates higher-level constraints while providing robust contingency plans that can accomplish the desired goals in a variety of situations[1]. Finally, the **arbitration level** includes higher-level functions such as goal planning, health monitoring, and user inputs. The arbi-

[1]Previous work by the authors provides one implementation of the first three levels of the proposed architecture [4].

tration level can be used to decide between accomplishing user-level demands as well as to pursue more undirected behavior designed to increase the robot's competency at a given task or in a given situation.

To implement the above architecture, the principles of component-based software design were used in order to facilitate the dynamic reconfigurability implicit at each level of the design. Implementing each block as a software component with a well-defined but generic interface, allows an autonomous system to easily swap in or out new sensory motor resources, controllers, and plans. In such a way, programming new capabilities into the system reduces to a resource allocation problem at the component level that can be submitted to machine learning algorithms to find fault-tolerant solutions, increase performance, and respond to previously unseen situations.

Specifically, the design is implemented using ROS [5] (for the planning and arbitration levels) and Orocos [6] (for the control and resource levels). This system will streamline R2 application development by leveraging the benefits of ROS, a service level architecture with tools for hardware abstraction, communication, package management, with the benefits of Orocos, a real-time core that has the ability to meet higher-performance requirements.

References

[1] M. Diftler, J. Mehling, M. Abdallah, N. Radford, L. Bridgwater, A. Sanders, R. Askew, D. Linn, J. Yamokoski, F. Permenter, B. Hargrave, R. Platt, R. Savely, and R. Ambrose, "The first humanoid robot in space," in *IEEE Conference on Robotics and Automation*, 2011.

[2] L. Sentis and O. Khatib, "Prioritized multi-objective dynamics and control of robots in human environments," in *Humanoid Robots, 2004 4th IEEE RAS International Conference on*, vol. 2, pp. 764–780, IEEE, 2004.

[3] R. Platt, M. Abdallah, and C. Wampler, "Multiple-priority impedance control," in *IEEE International Conference on Robotics and Automation*, 2011.

[4] S. Hart and R. Grupen, "Learning generalizable control programs," *IEEE Transactions on Autonomous Mental Development*, vol. 3, no. 3, pp. 216–231, 2011.

[5] "Robot Operating System." http://www.ros.org, November 2011.

[6] "The Orocos Project." http://www.orocos.org, November 2011.