Multi-Process Approach to Reliable Control of Humanoid Robots

Daniel M. Lofaro[1], Neil Dantam[2], Michael Grey[2], Mike Stilman[3], and Paul Oh[4]

***Abstract*—**

[1]

The Hubo2+ is a 1.3 (4' 3") tall, 42 kg (93 lb) full-size humanoid robot. The Hubo series was designed and constructed by the Korean Advanced Institute of Science and Technology (KAIST) and spinoff Rainbow Inc. [2]. It has 38 degree of freedom: six per arm and leg, five per hand, three in the neck, and one in the waist. Sensors include three-axis force-torque sensors in the wrists and ankles, accelerometers in the feet, and an inertial measurement unit (IMU). The sensors and embedded motor controllers are connected via a Controller Area Network to a pair of Intel Atom PC104+ PCs running a GNU/Linux distribution.

The Hubo2+ is a 1.3 tall, 42 full-size humanoid robot, produced by the Korean Advanced Institute of Science and Technology (KAIST) and spinoff company Rainbow Inc. [2]. It has 38 DOF: six per arm and leg, five per hand, three in the neck, and one in the waist. Sensors include three-axis force-torque sensors in the wrists and ankles, accelerometers in the feet, and an inertial measurement unit (IMU). The sensors and embedded motor controllers are connected via a Controller Area Network to a pair of Intel Atom PC104+ PCs.

Hubo-Ach [1] is an Ach-based interface to Hubo's sensors and motor controllers. This provides a conventional GNU/Linux programming environment, with the variety of tools available therein, for developing applications on the Hubo. It also efficiently links the embedded electronics and real-time control to popular frameworks for robotics software: ROS [3], OpenRAVE,[2] and MATLAB[3].

Reliability is a critical issue for software on the Hubo. As a bipedal robot, Hubo must constantly maintain dynamic balance; if the software fails, it will fall and break. A multi-process software design improves Hubo's reliability by isolating the critical balance code from other non-critical functions, such as control of the neck or arms. For the high-speed, low-latency communications and priority access to latest sensor feedback, Ach provides the underlying IPC.

Hubo-Ach handles CAN bus communication between the PC and embedded electronics. Because the motor controllers synchronize to the control period in a *phase lock loop* (PLL), the single `hubo-daemon` process runs at a fixed control rate and communicates on the bus. The embedded controllers lock to this rate and linearly interpolate between the commanded positions, providing smoother trajectories in the face of limited communication bandwidth. This communication process also avoids bus saturation; with CAN bandwidth of 1 Mbps and 200Hz control rate, `hubo-daemon` currently utilizes 78% of the bus. `Hubo-daemon` receives position targets from a `feedforward` channel and publishes sensor data to the `feedback` channel, providing the direct software interface to the embedded electronics.

Each Hubo-Ach controller is an independent processes. The controllers handle tasks such as balance, manipulation, and human-robot interaction. Each controller asynchronously reads state from the `feedback` Ach channel and sets reference positions in the `feedforward` channel. `Huobo-daemon` reads the most recent reference position from the `feedforward` channel on the the rising edge of its control cycle. This allows the controller processes to run at arbitrary rates without effecting the PLL of the embedded motor controllers or the CAN bus bandwidth utilization.

fig:huboachros shows an example control loop integrating Hubo-Ach and ROS. The `hubo-daemon` communicates with the embedded controllers at 200, publishing to the `feedback` channel. The `hubo-ach-ros` process bridges ROS topics and Ach channels. It translates messages on the `feedback` Ach channel to the `rFeedback` ROS topic and translates the `rFeedforward` ROS topic to the `ref` Ach channel. The `planner` process computes desired trajectories, which are relayed via `hubo-ach-ros` to `filter` for preprocessing to smooth the motion and reduce *jerk* before `hubo-daemon` communicates references to the embedded controllers. During operation, `rviz` displays a 3D model of the Hubo's current state. Each of these process runs asynchronously, communicating at different rates; however, `hubo-ach-daemon` maintains its 200 cycle, ensuring phase lock with the embedded controllers. This control loop effectively integrates real-time IPC and control under Hubo-Ach with the non-real-time ROS environment.

Hubo-Ach is in use for numerous projects at several research labs. Users include include groups at MIT, WPI, Ohio State, Purdue, Swarthmore College, Georgia Tech, and

*This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) award N65236-12-1-1005 for the DARPA Robotics Challenge.

[1]D. Lofaro is a Ph.D Candidate of the Electrical and Computer Engineering Department, Drexel University, 3141 Chestnut St, Philadelphia PA, USA `dan at danlofaro.com`

[2]N. Dantam and M. Grey are Ph.D students with the Department of Computer Science, Georgia Institute of Technology, Atlanta GA, USA `ntd at gatech.edu and mxgrey at gatech.edu`

[3]M. Stilman is with Faculty of Department of Computer Science, Georgia Institute of Technology, Atlanta GA, USA `mstilman at cc.gatech.edu`

[4]P. Oh is with Faculty of Mechanical Engineering, Drexel University, 3141 Chestnut St, Philadelphia PA, USA `paul at coe.drexel.edu`

[1]Available under permissive license, http://github.com/hubo/hubo-ach

[2]OpenRAVE: http://openrave.org/

[3]MATLAB: http://www.mathworks.com/

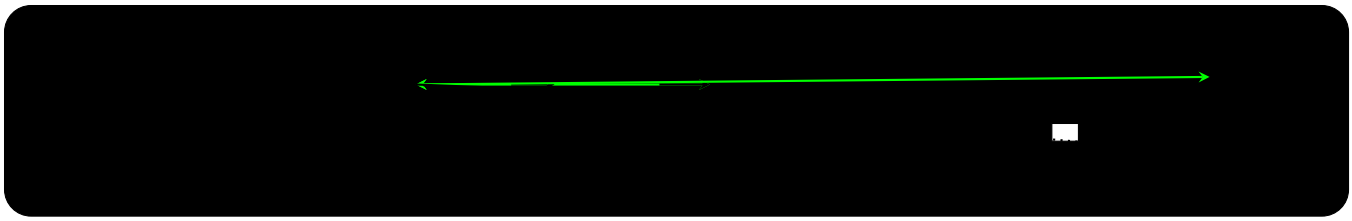

Fig. 1. Feedback loop integrating Hubo-Ach with ROS

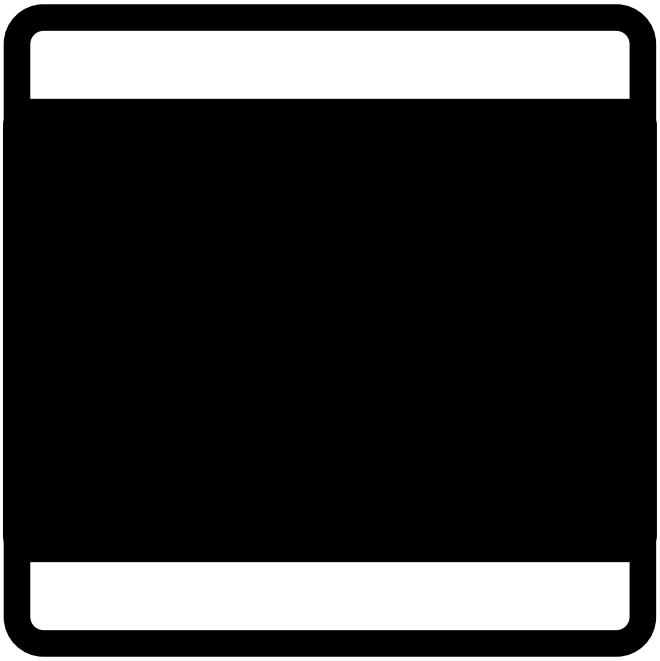

Fig. 2. Hubo (left) turning a valve via Hubo-Ach alongside Daniel M. Lofaro (right). Valve turning developed in conjunction with Dmitry Berenson at WPI for the DARPA Robotics Challenge.

Drexel University. These projects primarily revolve around the DARPA Robot Challenge (DRC)[4] team DRC-Hubo[5]. The DRC includes rough terrain walking, ladder climbing, valve turning, vehicle ingress/egress and more. fig:valve shows the Hubo using the Hubo-Ach system to turn a valve.

Hubo-Ach provides an effective base for developing real-time applications on the Hubo. Separating software modules into different processes increases system reliability. A failed process can be independently restarted, minimizing chance of damage to the robot. In addition, the controllers can run at fast rates because Ach provides high-speed low-latency communication with `hubo-daemon`. Hubo-Ach provides a C API that is easily called from high-level programming languages and integrates with popular platforms for robot software such as ROS and MATLAB, providing additional development flexibility. Hubo-Ach is a validated and easy to use interface between the mechatronics and the software control algorithms of the Hubo full-size humanoid robot.

[4]DARPA Robot Challenge: http://www.theroboticschallenge.org/
[5]DRC-Hubo Homepage: http://drc-hubo.com/

REFERENCES

[1] N. Dantam and M. Stilman, "Robust and efficient communication for real-time multi-process robot software," in *International Conference on Humanoid Robots (Humanoids)*, 2012.

[2] B.-K. Cho, S.-S. Park, and J. ho Oh, "Controllers for running in the humanoid robot, hubo," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, dec. 2009.

[3] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.