I. HUBO2 PLUS

Hubo2 Plus is a 130 cm (4' 3") tall, 42 kg (93 lb) full-size humanoid robot commonly refered to as Hubo. It was designed and constructed by Prof Jun-Ho Oh at the Hubo Lab in the Korean Advanced Institute of Science and Technology (KAIST)[1]. Hubo is anthropomorphic to a human meaning it has 2 arms, 2 legs and a head. There are 6 degreese of freedom (DOF) in each leg, 6 in each arm, 5 in each hand, 3 in the neck, and 1 in the waist; all totalling 38 DOF. All joints of the major joints are high gain PD position controlled with the exception of the fingers. The fingers are open-loop PWM controlled. The sensing capiability consists of a three axis force-torque (FT) sensor on each leg between the end of the ankle and the foot as well as between where the arm connects to the hand. Additionally it has an inertial measurement unit (IMU) at the center of mass and accelerometers on each foot. The reference commands for all of the joints are sent from the primary control computer (x86) to the individual motor controllers via two Controller Area Network (CAN) buses. There are currently eight Hubo's functioning in the United States as of December 2012. Four reside at Drexel University and one at Georgia Tech, Perdue, Ohio State and MIT. Jaemi Hubo is the oldest of the Hubos in America and has been at the Drexel Autonomous Systems Lab[1] (DASL) since 2008[2]. Fig. 1 shows the major dimensions of Hubo.

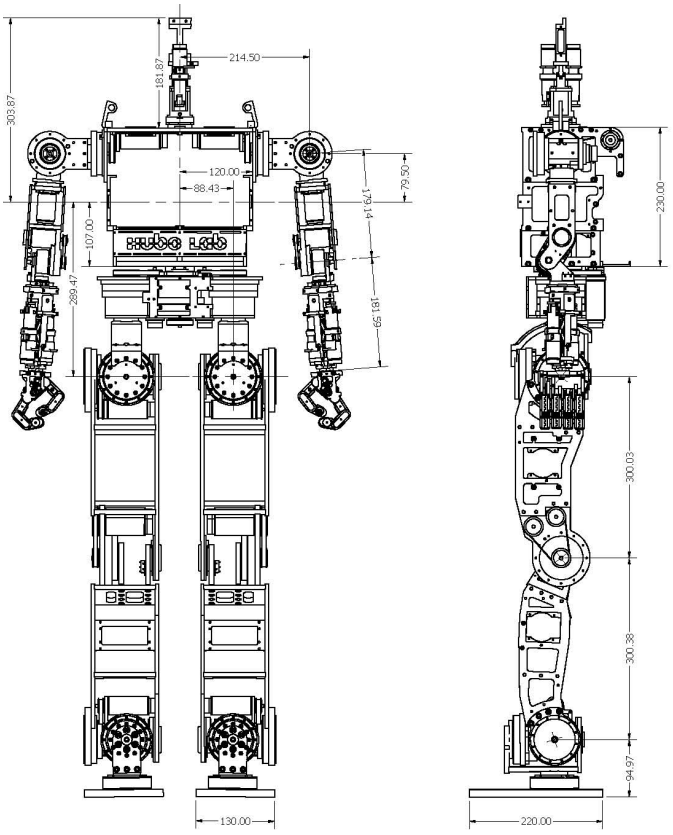

Fig. 1. Hubo2 Plus platform: 38 DOF, 130 cm tall full-size humanoid robot weighing 37 kg.

[1]Drexel Autonomous Systems Lab: http://dasl.mem.drexel.edu/

II. HUBO-ACH: LINUX ON HUBO

Hubo-Ach is the open-source, Linux based, BSD licensed system run on Hubo. It was designed by Daniel M. Lofaro[2] and Neil Dantam and created in collaboration with the *Drexel Autonomous Systems Lab* at Drexel University and *Golems - The Humanoid Robotics Laboratory*[3] at the Georgia Institute of Technology.

The overarching goal of the Hubo-Ach system is to create an easy to use interface between the Hubo's hardware and the programming environment. System design decisions were made with the programmers and developers of the Hubo in mind. This design philosophy streamlines closed-loop controller implementation, human robot interaction development and the utilization of popular robot related systems such as ROS[4] (Robot Operating System), OpenRAVE[5] and MATLAB[6] on the Hubo platform.

Hubo-Ach is a daemon process that uses a high-speed, low-latency IPC called Ach [3] to comunicate with controllers. Each controller are indipendent processies. It is designed such that controllers are able to command each joint over an Ach channel at arbratary rates. Hubo-Ach impliments a real-time loop in which all of the motor references and state data are sest and updated respectively. The motor references are set to the Hubo over the CAN bus with the latest reference on the feedforward chanel. This is sent at the rising edge of the real-time loop. Sensor updates are requested directly after all references are sent. The real-time loop runs with a period of T_0. T_0 is currently set to 0.005 $sec.$ This causes a CAN bus utiliztion of 78%. The real-time loop in Hubo-Ach is needed to ensure the internal phase lock loop (PLL) of the motor controllers lock onto the reference update rate. This is then used for linear interpolation of the reference onboard the motor controllers. The real-time loop is also is used to ensure the CAN bus bandwidth is not saturated.

Fig. 2 shows an example of how the Hubo-Ach system works. *Process 1* gets feedback from Hubo via the **Feedback** Ach channel populated via Hubo-Ach. *Process 1* requests feedback data with a period of T_1. If $T_1 > T_0$ then *Process 1* will receive the most recent state data received by Hubo-Ach from the Hubo with no overlapping frames. If $T_1 < T_0$ *Process 1* will also receive the latest state data received by Hubo-Ach from the Hubo however some frames will overlap causing identical state data to be returned. *Process 1* performs calculations for the given task or control. The resulting reference commands are created and sent to *Process 2* via the **Ref** Ach channel with an update period of T_2. *Process 2* is a low-pass filter that limits the jerk on each joint. The resulting trajectory is sent to the **Feedforward** Ach chanel with a period of T_3. T_3 is currently set to 0.01 $sec.$ This means that T_2 does not have to be a regular rate. Thus it is possiable for $T_2 >> T3$ with out harm to the robot even it it is a step input. The resulting movements of Hubo are smooth and jitter free.

[2]Daniel M. Lofaro: http://danlofaro.com/
[3]Golems - The Humanoid Robotics Labatory: www.golems.org/
[4]ROS: http://www.ros.org/
[5]OpenRAVE: http://openrave.org/
[6]MATLAB: http://www.mathworks.com/

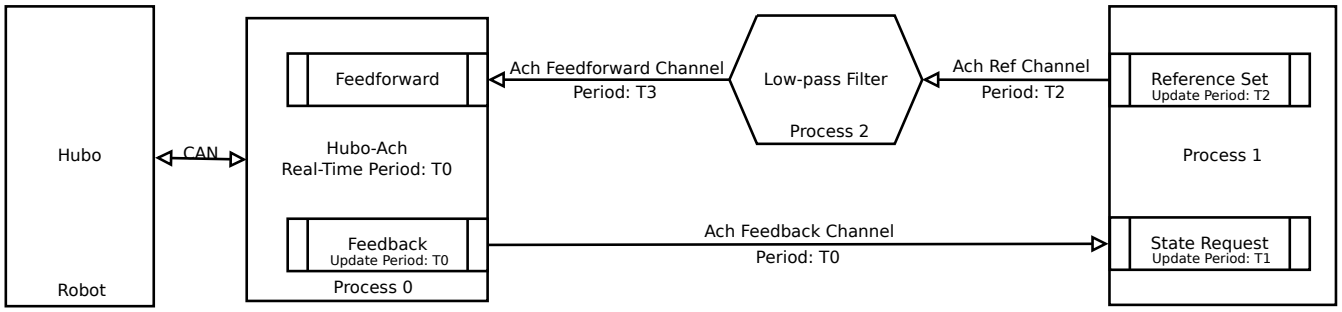

Fig. 2. Hubo-Ach interfacing with two seperate processes creating a closed loop system. Process 0 (Hubo-Ach) takes the most recent command from the Feedforward Ach channel. It sends the reference over the CAN bus to the Hubo. The state of the robot is updated and stored in the Feedback Ach chanel. Process 1 reads the state at its own rate, proforms the desired control then sends the the desired references to Process 2. This process is a low-pass filter that reduces the jerk on each joint. It smooths the desired trajectory allowing Process 1 to send step input to any of the joints at any rate. The reference command for each joint is set on the Feedforward Ach channel at a regular rate compleating the closed loop system.

The key point is that Hubo-Ach updates the state data in the **Feedback** Ach channel commands the motors with the references from the **Feedforward** Ach channel in real-time with a preiod of T_0 no matter what rate the external controller is updating the feedforward channel. In addition because each controller is its own process, if one of the processes crashes Hubo-Ach does not. The failed process can then be restarted with no harm done to the robot.

REFERENCES

[1] Baek-Kyu Cho, Sang-Sin Park, and Jun ho Oh. Controllers for running in the humanoid robot, hubo. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, dec. 2009.

[2] Daniel M. Lofaro, Robert Ellenberg, Paul Oh, and Jun-Ho Oh. Humanoid throwing: Design of collision-free trajectories with sparse reachable maps. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, oct. 2012.

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