A Common Interface for Humanoid Simulation and Hardware

Robert Ellenberg, Robert Sherbert, Paul Y. Oh, Alex Alspach, Roy J. Gross, JunHo Oh
Drexel University, Philadelphia, PA, 19104
Korea Advanced Institute of Science and Technology, Daejeon, SK
rwe24@drexel.edu, rs429@drexel.edu, paul@coe.drexel.edu,
ana35@drexel.edu, rjg48@drexel.edu, jhoh@kaist.ac.kr

***Abstract*— Humanoid robotics development often depends on simulation and prototypes for new walking algorithms. The advantages of simulation such as low cost and risk make repeated experiments and development more straightforward. However, real world implementation can require extensive porting to move from a simulation environment to a hardware platform. This tedious task can be a huge time sink that drastically slows development cycles. This paper describes a way to eliminate this bottleneck. By developing standardized protocols for motors and sensors, a software controller can communicate with both platforms using the same interface.**

I. Introduction

For the last few years, the state of the art in humanoid research has been full-size humanoids such as Honda's ASIMO[1], KAIST's Hubo [2], and the HRP-2[3]. These robots are capable of subtle, sophisticated motions that, in many cases, equals the dexterity of humans. These robots can interact with humans using vision, speech recognition, gesture recognition, and even touch sensing.

Unfortunately, these capable tools are very difficult for smaller labs to acquire and use. Large humanoids are expensive (on the order of 500 000 USD) and delicate custom-built machines. They require careful, expert assembly and maintenance to produce consistent results. A humanoid's weight and strength means that safety equipment is often necessary, and falls can be expensive to repair. The cost and complexity of these machines creates high barriers to entry into the humanoids field, and effectively retards development by discouraging talented researchers who simply do not have access to the hardware.

The obvious solution to the problems faced by humanoids (expense and complexity) is to attempt alternative approaches to the full scale robots. Engineers commonly use the techniques of both scaling and simulation to address problems of this nature.

Scaling has already been established in the literature and culture of the humanoids community. This suggests that miniature humanoids provide a valuable platform for certain topics. The stated goal of RoboCup is to field a competitive artificial soccer team by 2050[4]. Towards that end, the competition includes a miniature class to cut development costs for competitors. Similar competitions such as Robogames and RoboOne also value the miniature humanoid, as both a stepping stone and a source of insight into problems such as obstacle avoidance[5].

Simulation eliminates the risks and costs of working directly with a humanoid. Virtual environments are easily shared and duplicated, making it easy to collaborate with distant researchers on complex problems. Popular simulation packages such as OpenHRP3[6], USARsim[7], and Webots [8] are powerful tools for prototyping walking gaits and gestures. They can simulate rigid body dynamics, collisions, and sensors.

While scaling and simulation can be used to guide research, they can never prove the reliability of an approach when implemented on a full-sized robot. The only certain verification is to take the techniques developed using alternative methods and apply them to the full scale machine. In doing this, however, one of the main benefits of the alternative methods, speed, is lost. The software architectures of the three approaches, along with important intermediaries such as trajectory generation and inverse kinematics, differ widely. To move from one system to another under present conditions can require as much as, if not more, time than researching the original question.

To address the problem, this paper details efforts to create a platform and tool set which will allow researchers seamless transitions between the three approaches. The platform consists of an open set of hardware and software humanoids (full-sized Hubo, scaled Mini-Hubo, and simulated Virtual Hubo). The tools include controller software that is portable between the three, scalable trajectory generation, and flexible inverse kinematics.

The open humanoids architecture, detailed in Section II, describes a three-pronged approach to the issue of hardware accessibility. The first mode of access introduces a system for researchers to remotely experiments on a full-sized humanoid robot. The second is an open design for a low-cost miniature humanoid that includes physical and production specifications. The third is a humanoid simulation platform that includes full models of the full and miniature robots.

Among the included tools in this effort is a software framework, detailed in Section III, which attempts to address

This work was funded through the National Science Foundation's PIRE: Humanoids Grant No. 0730206 and the NSF Career Grant No. IIS-0644151. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

the issue of controller portability across the platforms. The work examines presenting the hardware to the programmer in something akin to the state space approach. This contrasts with many hardware abstraction systems in robotics, which tend to abstract hardware at the behavioral level (e.g. 'move forward', 'turn left'). Section IV explains the design of a scalable Zero Moment Point walking controller based on established methods. Section V rounds out the work by describing an inverse kinematics engine that can handle variations in geometry and joint layout between the various humanoids used in the system.

II. Supporting Hardware

A. Full-Sized: Hubo

The primary goal of this work is to make the transitions between the various forms of humanoid robots as seamless as possible. An additional aim is to make experimentation with any single platform maximally efficient. For full-sized humanoids, the primary hindrance to experimentation is their own size and delicacy. Slips and falls can cause significant damage, taking many hours to return the robot to a usable state. Even with an ideal outcome, the robot must still be manually reset for subsequent experiments. Experience with Drexel's Jaemi Hubo, a version of the KAIST humanoid, supports this notion.

In order to resolve this issue, an automated safety harness was designed to follow the robot without interfering directly with its motion. The harness follows the Hubo by measuring the location of the Hubo's head and moving itself such that it is positioned to catch the Hubo should a fall occur, while not interfering with the operation of the system otherwise. (Figure 1).

The harness has been implemented within a room-sized 3 Degree of Freedom (DOF) gantry robot as described in [9]. The end effector of this robot can move within a 6x6x3 meter enclosure. The enclosed environment provides an isolated space for experiments, allowing operators to customize environmental conditions such as lighting. With the harness attached to the end-effector, the Hubo can freely walk within this area at much-reduced risk.

To sense the head's motion, a small thread is connected between the gantry arm and the Hubo's head. As the head moves, it gently pulls the thread to one side (Figure 2). This thread passes through a tube attached to a small gimbal, and is gently pulled taut with a light return spring. The pull due to movement tips the gimbal arm, measuring the angles with respect to the ground normal. By measuring the extension of the thread, and decoupling the measurements, an estimate of the head's displacement is possible. The sensor's low force actuation and mechanical connection ensure that the position is always measured, but with a minimum of disturbance to the robot's balance.

Knowing the displacement of the sensor thread, and the angles of the thread with respect to each vertical plane, the true head displacement is found by simple geometry (1). The 2D displacement vector $\vec{r}$ can be derived from the sensor using this formulation,

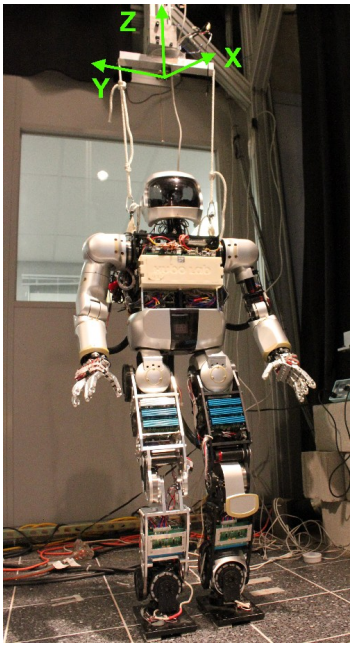

Fig. 1. Jaemi Hubo supported by safety harness connected at the shoulders. A small thread is attached to the head, between the harness ropes. The motion of this thread is measured to track the head displacement, which allows the harness to follow, keeping the ropes slack.

$$\vec{r} = (z_0 + \Delta z) \begin{pmatrix} sin(K_v v_x) \\ sin(K_v v_y) \end{pmatrix} \quad (1)$$

where x is a horizontal displacement component, Δz is the change in length of the sensor string, and K_v is a constant converting sensed analog voltage to sensor angle.

Similarly, the displacement in the Z direction is given by (2), where Z is the actual distance in the z-axis from the top of the head to the harness base. This feedback, along with a simple PID controller, causes the harness to follow the head at a safe distance. The controller's gains are tuned for a slow, damped response, so that the harness does not track small changes in head motion.

$$Z = \sqrt{(z + z_0)^2 - r^2} \quad (2)$$

Crash detection is an important feature of the harness, so that it does not simply follow the robot in a fall to the ground. During normal operation, tracking velocity is limited by the controller to a safe bound. If the robot falls, it will exceed the tracking speed of the harness. This sudden motion will pull the support ropes tight, while the load is absorbed by a soft spring. The compression also triggers a switch, indicating a crash. Now the software can intervene, pausing the experiment and lifting the Hubo to safety.

The gantry includes power and network connections to the Hubo to help automate other experimental chores. A combination of external cameras and logging of the rrobot's internal communication gives a complete picture of its current state. Our hope is that these features will largely eliminate human intervention, allowing safe remote operation with minimal local supervision.

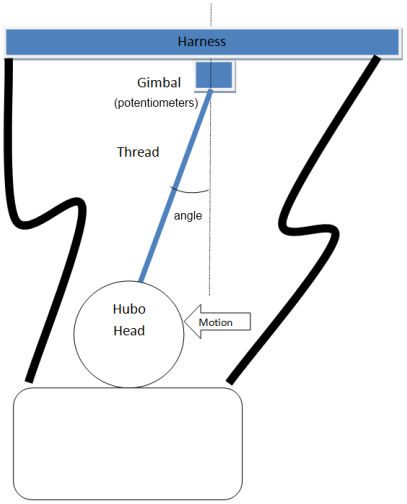

Fig. 2. Hubo harness drawing, showing the angle the thread makes as the head moves side to side. A n arm mounted on a joystick gimbal senses this angle.

B. Scaled Size: Mini-Hubo

In addition to providing easier access to the full scale robot, this work also seeks to provide a ready toolkit for experimentation with miniature humanoids. Miniatures have the advantage that they are far less expensive than their full sized counterparts (systems commonly sell at near the 2000 USD price point). This allows even small laboratories to obtain and experiment with the systems.

Towards this end, our team, lead by collaborators at Virginia Tech [] have developed a 'Mini-Hubo' which has a joint layout that mirrors the layout of the full sized version. Roll/pitch/yaw order is identical for a given joint, though the limb lengths and motion range vary between the two. The end effect of this is that only the transformation matrices involved in inverse kinematics change, while the form of the equations remains the same.

While working with such a scaled system can save significant effort, the relationship between platforms becomes complicated. Key scaling issues include:

- Kinematics: Limb length and joint arrangement
- Dynamics: moments of inertia and mass differences
- Mechanical strength and stiffness
- Actuator power-to-weight ratios

C. Simulation: Virtual Hubo

A third approach towards increasing the accessibility of humanoids platforms is in providing a simulation environment for both the miniature and full sized robots described earlier. Physical models which account for the robot's full geometry, mass, and inertial properties have been created using the Webots (which back-ends to the Open Dynamics Engine) simulation environment.

The complexity of environments that can be simulated is limited. Common simplifications are to assume rigid bodies, flat surfaces, and ideal servo motors. Tasks such as rough-terrain walking may need extensive testing with a real robot to produce accurate results. After this investment, however, simulations provide repeatablity and reproducibility not possible with hardware. They can also give a rapid assessment of the effectiveness of a base gait to check for things like physical interference before moving on to the actual hardware.

III. Addressing Software Portability

The bridge between the three different approaches (full scale, miniature, and virtual) to humanoid development stands within the software that runs each. It is obvious that the same driving mathematics underlies each approach, but if the researcher cannot move these equations seamlessly between the three platforms, he/she is effectively crippled. It is easy, when using such complex systems, to spend more time implementing motor drivers, sensor interfaces, and real-time subsystems, than on the actual algorithms to be tested.

To fix this problem, this work includes a common software interface to the three methods of development. Sadly, standard packages such as Player/Stage[10], Microsoft Robotics Studio, ROS (the Robot Operating System)[11], etc. would only be marginally useful under these circumstances. These robot development environments are mostly built to work around the robot as a black box. They are targeted at implementing complex but high-level algorithms such as vision processing, navigation, path planning, and human interaction. Because of this focus, the environments provide little assistance in coordinating physical motions.

The greatest shortcoming of modern robotic development environments is a misconception about what is an atomic operation for a given robotic system. Current development environments approach the problem by asking a designer to implement the mechanics needed to perform high level behaviors, such as 'take a step', 'turn 20° right', or 'accelerate at $1\frac{m}{s}$'. This approach is adaquate for statically stable robots (e.g. tracked or four wheeled vehicles), but has little meaning for robots which are statically unstable but can dynamically stabilized. Designs involving legged robots such as humanoids, quadrupeds, or hexapods have contextual meanings for such high level operations. In these systems, a careful orchestration of actuators must be made in order to maintain stability. The interplay between high level commands (step forward) and the actuation pattern needed to carry them out is not always obvious. An excellent example of this situation is the BigDog[12] robot, where the rough terrain environments the robot is designed to handle means that an instruction like 'forward' has very different actuator interpretations depending on context. Fortunately, resolving this problem requires only a simple shift in thinking about the system.

A. State Space Inspired System Presentation

The solution involves abstracting the hardware of the system in terms of the common mathematical representations used to model robots. In other words, the key to moving

seamlessly between the three humanoid platforms is expressing the controllers, in code, in a format similar to the state space representation. It is a way of presenting the low level hardware (sensors/actuators) in terms of these states.

The approach is based on providing the user (the programmer who is implementing the control algorithm) with access to the physical properties of the robot in terms of the state variables. Each state variable of the robot (e.g. elbow position, torque at the elbow motor, etc) can be accessed through a programmatic variable which can provide history, the present state, and future projections of the state. This allows the controller to be expressed in terms of the more theoretically common state-space approach, as opposed to the common procedural programming constructs.

The mechanisms which enable this representation are somewhat complex, but worthwhile to achieve the state variable representation to the end user. Since the entire system is built within the context of a real-time operating system, each distinct component is represented as a unique task. A task is associated with its own thread, which can be run in parallel with other tasks' threads, and possesses its own stack space (memory storage area). The task-based design is an inherently flat, non-hierarchical one. While a layered structure is introduced in describing the system, each task is capable of bypassing this structure to deal with exceptions - providing flexibility to the design. The layers of the design are as follows: state-variable access layer, communications protocol layer, and a hardware gateway layer.

The uppermost layer of the program is the portion available to the author of the control system: the state variables. The operations of interest for a state are reading, writing, and prediction. A state is initialized with a data structure that provides the identifying information of the underlying hardware. This could be hardware identification numbers, Internet addressing numbers, etc. Using this identifying information the state can make requests to the hardware, through the protocol, in order to sample the state (read) or alter the state (write). The state variable is also designed to automatically store a history of previously sampled values, so they can be used for averaging or predictive algorithms.

The next layer is the communications protocol layer. When each state is created, it is registered with a protocol instance based on the type of hardware it is communicating with. This protocol understands how a data stream must be constructed to communicate with the physical hardware that will provide the requested information. Since multiple states can be associated with the same physical hardware (e.g. a motor torque state and a motor rotational velocity state both exist for a single motor) the protocol is designed to be capable of accepting requests from different states and combine them into efficient (in terms of bus usage) requests to the hardware.

The protocol itself receives a reference to a hardware communication object which serves as a gatekeeper to the physical bus. Since the design of the system is real-time and task based, and therefore tasks can interrupt one another, allowing multiple protocols direct access to the bus could result in scrambled, intermixed requests on the bus. By buffering and funneling these requests through a gatekeeper task, the design assures that messages cannot become scrambled.

B. Application to Humanoids

how this system adapts itself to humanoid robotics is the next obvious question to address. The user (who is the controller designer) initializes a number of state variables, each one corresponding to the degrees of freedom of the machine. For the case of a humanoid robot these would include states for each of the motor position and/or velocity states, the motor currents (equivalently torques), the angular and linear rates associated with any inertial measurement units, etc.

Each state will be tied to an appropriate protocol which will handle its communication with the actual hardware. For the motor related states: Since all motor controller hardware will likely be located on the same electrical bus all these states will reference the same protocol. This protocol will handle issuing the appropriate byte-stream to the hardware to make the requests as well as condensing the requests to facilitate efficient usage of the bus. The protocol passes the byte stream it design along to a hardware object, which acts as a gatekeeper for the actual physical bus.

The described system lends itself well to scaling in humanoid robotic applications. The user will write any upper level functions based on the state variables. He has the flexibility to write controllers directly in terms of his state variables, or can generate any amount of higher level controller based on these state variables. In scaling the humanoid platforms, the hardware involved (motors, inertial measurement units, etc) is changed but the states involved remain virtually unchanged.

The joint layout of the robot remains nearly identical, and thereby allows the representation at the state layer (and therefore any controllers or infrastructure built on top of it) to be maintained. The hardware instances change, but only within a limited range. Since most hardware devices will enumerate within the operating system as character devices, few changes are required to port this level of the control. The only major change which must be made when shifting between humanoid platforms is the protocol layer, which must be redesigned to support manufacturer specifications for the particular motor used.

IV. Trajectory Generation

At a high level, similarities between robot platforms and simulation structure can be advantageous. Despite gross differences in mass and size, for example, both Mini-Hubo[13] and Jaemi Hubo[14] can perform simple dynamic walking. A popular method, the Zero Moment Point (ZMP)-based algorithms used for these robots has a simplified model of the humanoid body, bypassing many of the scaling issues identified. In the simple point-mass, inverted pendulum model used in [2], the only parameters are the overall robot mass, and the height of the center of gravity. Scaling this algorithm

is very direct because of its simplicity, but it does require tuning to compensate for the simplifying assumptions.

Because of the simplifying assumption that robot body behaves as an inverted pendulum, the moment of inertia and masses of the legs are not modeled. While the effects of the leg swing on the body motion are real and require tuning and/or feedback control to eliminate, it is beyond the scope of this project. Our initial strategy designs out the problem with a slower walk and lower posture to improve stability.

One walking cycle consists of a pair of steps of arbitrary length. The apparent step length is specified for each step. If the step had started from ideal posture, the foot would land such that it was $\frac{1}{2}$ of the step distance past the hip at the end of the step. The trajectory generator subtracts the difference between the actual and ideal initial pose to produce this outcome.

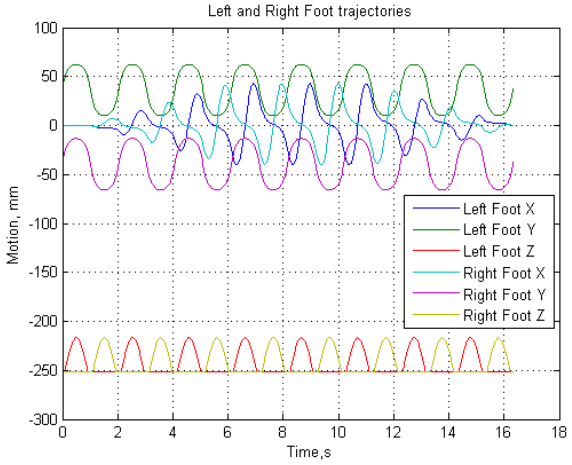

Fig. 3. Foot trajectories relative to the hip motion, showing alternate lift & landing using sinusoidal (Z) and cycloidal (X) motion.

In general, turning while walking forward introduces a centripetal acceleration component, which in turn affects the zero moment point location. While it is possible to reformulate the ZMP equation with additional accelerations factored in, a first approximation was developed using a simple hip yaw offset.

To complete a turn, two footsteps are required. The first step leads the turn, which requires that a left turn begin with a left step, and vice-versa. In the first step, each hip yaw joint's yaw constraint is given an opposite offset, following a cycloidal amplitude. The following step inverts this path, bringing both legs' yaw angles back to zero (Fig. 4). These angles are specified for the solve as the Y-components of the foot's X vector in the global frame.

V. Inverse Kinematics Implementation

A trajectory calculated for a full-scale humanoid robot must be adjusted to fit the range of motion of a miniature humanoid robot. Varying limb reach and collision boundaries also affect the range of achievable motions. To effectively move across platforms, these limits must be parametrized

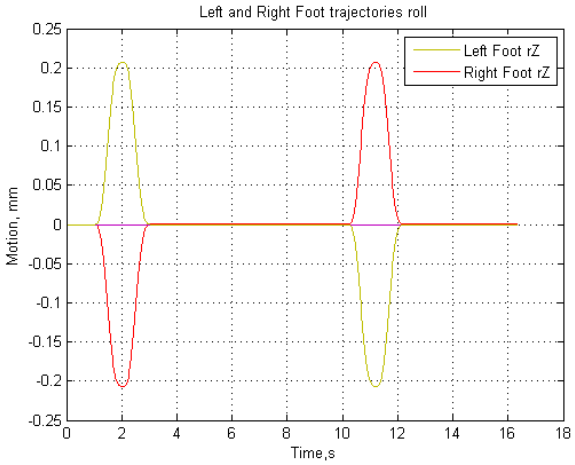

Fig. 4. Simple hip yaw offsets for low-velocity turning

such that they are either compensated for in the design of the trajectory, or online using a learning algorithm.

Ideally, the inverse kinematics(IK) solution is transparent to the end user. In the proposed software framework, the state handler automatically runs the IK to translate from cartesian space to joint space. When polling sensed foot position, the corresponding forward kinematics can be called as part of the handler to automatically produce the data from the measured joint angles. Unfortunately, these functions are specific to the kinematics of each robot. To quickly develop the necessary solver functions, a MATLAB program was developed to use robot dimensions and joint order to generate a custom IK solver as C code.

To prototype the solution method, an inverse kinematics solver based on the pseudoinverse and selectively damped least squares methods in [15] was created in MATLAB. The symbolic math toolbox is used to create the constraint equations in symbolic form. The equations in general are functions of the joint angles and link parameters. The solution of the least squares model uses the singular value decomposition of the Jacobian to simplify the computation. A common method for calculating the least squares estimate simplifies the actual inverse by inverting only the S matrix.

The general skeleton of the mini-hubo and Jaemi Hubo has 6 DOF in each leg, with the order of rotation joints common to both (Fig. 5). In the hips, ankles and shoulders, multiple joint axes intersect at a point. The rotation of the first link of each limb can be contained on a spherical surface about this point. With the robot's torso as a frame of reference, there are 4 kinematic chains to be solved. Full specification of leg motion requires each foot's position in space (3 constraints), plus the foot's orientation in space (3 constraints). The arms each need a space constraint at the wrist. This gives a total size of the Jacobian of 18x21.

Once the torso's position in space is specified, the IK problem could be solved independently for each limb with respect to the torso. While this approach is efficient (no Jacobian is larger than 6x6), it does not optimize the torso

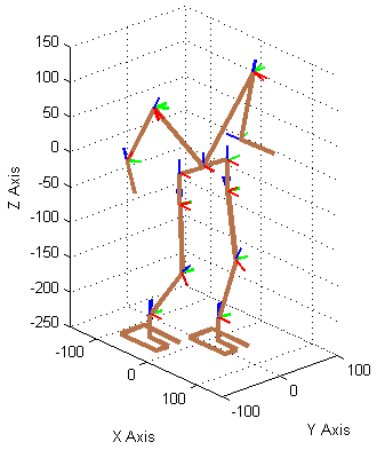

Fig. 5. The kinematic skeleton of the Mini-Hubo robot, used to implement the inverse kinematics solver. Shown are 6 joints each for the legs, the hip, and 4 joints for each arm.

rotation with respect to the hip. If a large step is needed, for instance, rotating the torso yaw joint aligns the hips with the direction of the step. The resulting Jacobian matrix thus takes the block form of (3).

$$J_{robot} = \begin{pmatrix} J_{legs,12x13} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & J_{Larm,3x4} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & J_{Rarm,3x4} \end{pmatrix} \quad (3)$$

An important consequence of this solution method is that the pseudoinverse does not cause any first-order motion along the nullspace of J [15]. For under-constrained systems, the nullspace is not empty. The matrix given by (4) projects any vector onto the nullspace of J, giving $\vec{\phi}$. Practically, this means that the joint space solution can be influenced by the addition of an arbitrary influence vector.

$$\vec{\phi} = [\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J}]\psi \quad (4)$$

VI. Conclusion

The gaps between humanoid robots and their simulations currently require a significant effort to compare results in a meaningful way. The software structure currently in development will eliminate this delay, allowing both simulation and hardware to be refined simultaneously. By simplifying the porting of humanoid control and algorithms, the framework proposed will allow fresh collaboration between otherwise alienated researchers and robots.

The biggest outstanding goal of the system is to complete the abstraction layer between the hardware bus and the software controller. Several technical hurdles must be overcome for this to be possible. As more humanoids are added the the system, each new computer architecture and communication bus will require it's own set of protocols. While this development burden is significant, it is more akin to authoring a device driver rather than an operating system. The standard framework in which to build the protocols for the hardware will hopefully provide a significant time savings in implementation. The utility of the entire system will grow as new protocols are added, multiplying the value of each future contributors' work.

References

[1] Honda, "Honda develops intelligence technologies enabling multiple asimo robots to work together in coordination," Press Release, December 2007.

[2] Park, I.-W., Kim, Y.-D., Lee, B.-J., Yoo, J.-K., and Kim, J.-H., "Generating performance motions of humanoid robot for entertainment," *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on*, pp. 950–955, Aug. 2007.

[3] Kankeo, K., Kanehiro, F., Kajita, S., Hirukawa, H., Kawasaki, T., Hirata, M., akachi, K., and Isozumi, T., "Humanoid robot hrp-2," in *Proceedings of the 2004 IEEE International Convention on Robotics & Automation*. New Orleans: IEEE, April 2004.

[4] Cheng, X., Zhang, J., and Ke, J., "Robocup is a stage which impulse the research of basic technology in robot," in *Robotics and Biomimetics, 2006. ROBIO '06. IEEE International Conference on*, 17-20 2006, pp. 965 –970.

[5] Wong, C.-C., Cheng, C.-T., and Huang, K.-H., *Design and Implementation of Humanoid Robot for Obstacle Avoidance*, Papers Presented at Congress, Robogames 2007 Std., 2007.

[6] Kanehiro, F., Fujiwara, K., Kajita, S., Yokoi, K., Kaneko, K., Hirukawa, H., Nakamura, Y., and Yamane, K., "Open architecture humanoid robotics platform," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 1, 2002, pp. 24 – 30 vol.1.

[7] Wang, J., Lewis, M., and Hughes, S., "Validating usarsim for use in hri research," in *Proceedings of the Human Factors and Ergonomics Society 49th Annual Meeting*, 2005.

[8] Michel, O., "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004. [Online]. Available: http://www.ars-journal.com/International-Journal-of- Advanced-Robotic-Systems/Volume-1/39-42.pdf

[9] Narli, V. and Oh, P., "Hardware-in-the-loop test rig for designing near-earth aerial robotics," *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2509–2514, May 2006.

[10] Collett, T. H. J. and Macdonald, B. A., "Player 2.0: Toward a practical robot programming framework," in *in Proc. of the Australasian Conference on Robotics and Automation (ACRA*, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.118.6143

[11] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y., "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[12] Raibert, M., Blankespoor, K., Nelson, G., and Playter, R., "Bigdog, the rough-terrain quadruped robot," in *Proceedings of the 17th World Congress: The International Federation of Automatic Control*, 2008.

[13] Jun, Y., Ellenberg, R., and Oh, P. Y., "Realization of miniature humanoid for obstacle avoidance with real-time zmp preview control used for full-sized humanoid," in *Proceedings of the 10th IEEE-RAS International Conference on Humanoid Robotics*, 2010, unpublished Document.

[14] Oh, J.-H., Hanson, D., Kim, W.-S., Han, I.-Y., Kim, J., and Park, I.-W., "Design of android type humanoid robot albert hubo," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct. 2006, pp. 1428–1433.

[15] Buss, S. R., "Introduction to inverse kinematics with jacobian transpose, pseudoinverse, and damped least squares methods," University of San Diego, Department of Mathematics, Tech. Rep., 2009.