# A general-purpose system for teleoperation of the DRC-HUBO humanoid robot

Matt Zucker[1], Sungmoon Joo[2], Michael Grey[2],
Christopher Rasmussen[3], Michael Stilman[2], Aaron Bobick[2]

[1]Department of Engineering
Swarthmore College
Swarthmore, PA
*mzucker1@swarthmore.edu*

[2]School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA
*sungmoon.joo@cc.gatech.edu*
*mxgrey@gatech.edu*
{*mstilman, afb*}*@cc.gatech.edu*

[3]Department of CIS
University of Delaware
Newark, DE
*cer@cis.udel.edu*

March 1, 2014

## Abstract

We present a general system focused on addressing three events of the 2013 DARPA Robotics Challenge trials: debris clearing, door opening, and wall breaking. Our hardware platform is DRC-HUBO, a redesigned model of the HUBO2+ humanoid robot developed by KAIST and Rainbow, Inc. Our system allowed a trio of operators to coordinate a 32 degree-of-freedom robot on a variety of complex mobile manipulation tasks using a single, unified approach. In addition to descriptions of the hardware and software, and results as deployed on the DRC-HUBO platform, we present some qualitative analysis of lessons learned from this demanding and difficult challenge. Topics of particular interest include the critical roles of systems engineering and task allocation within a large robotics team, as well as strategies for risk management and testing.

## 1 Introduction

Previous DARPA-sponsored competition programs in robotics have promoted research in topics as diverse as learning for ground vehicles [20], legged locomotion [32], and

autonomous manipulation [19]. Of all of these programs, however, the 2013 DARPA Robotics Challenge (DRC) is most similar in scope to the 2004-2005 Grand Challenge and the 2007 Urban Challenge [7, 8], which helped bring about a revolution in the field of autonomous driving. Like the driving challenges, the DRC is a broadly scoped, and aimed at producing robotic systems which integrate expertise from nearly every subdiscipline of robotics. We see these "Grand Challenge"-style competitions as valuable opportunities to apply both established research methods and techniques as well as novel ones to scenarios outside of carefully controlled laboratory conditions.

In this paper, we document a general-purpose system aimed at addressing three events of the DRC with the DRC-HUBO robot: debris removal, door opening, and wall breaking. Notably, our system allows a small number of human operators to teleoperate a 32 degree-of-freedom humanoid robot to perform a wide variety of tasks in the face of high communications latency and low bandwidth. Our high-level approach achieves generality in part by minimizing the amount of task-specific knowledge embedded in the system, instead allowing operators to coordinate high-level behavior through interpolation between key poses of the robot, respecting balance and pose constraints.

We note that the authors of this paper form just a small subteam of the much larger DRC-HUBO team, which was led by Dr. Paul Oh of Drexel University. Team DRC-HUBO began from a nucleus of institutions involved in the *Unifying Humanoids Research* NSF MRI-R2 grant and eventually grew to encompass researchers from Columbia University, the University of Delaware, Georgia Institute of Technology, Indiana University, KAIST, the Ohio State University, Purdue University, Swarthmore College, and Worcester Polytechnic Institute. Due to the large, distributed nature of the team, the participating institutions each formed subteams focused on one or more individual DRC events, with the authors' subteam responsible for the three events listed above. For information about our colleagues' approaches, we refer the reader to their existing publications related to the DRC [4, 12, 35, 36].

The remainder of this paper is organized as follows: in Section 2 and Section 3, we provide an overview of the system hardware and software, respectively. In Section 4, we document the results of trials at the DRC as well as results of our own in-house testing. Qualitative analysis of some key failures and successes are presented in Section 5. Finally, we conclude in Section 6.

# 2 Hardware

DRC-HUBO's hardware design is based on its immediate predecessor, HUBO2+, developed at KAIST [36]. The first generation of HUBO (KHR-1) debuted in 2002, and there

Table 1: Development of the HUBO series.

| Model | KHR-1 (2002) | KHR-2 (2004) | KHR-3/ HUBO (2005) | Albert HUBO (2005) | KHR-4/ HUBO2 (2008) | HUBO2+ (2011) | DRC-HUBO (2013) |
|---|---|---|---|---|---|---|---|
| Weight | 48 kg | 56 kg | 55 kg | 57 kg | 45 kg | 43 kg | 52 kg |
| Height | 120 cm | 120 cm | 125 cm | 137 cm | 125 cm | 130 cm | 147 cm |

have been several platform upgrades since then [22,23,26,27,29,30,36]. The development of the HUBO series is summarized in Table 1. The design of the upgrade from HUBO2+ to DRC-HUBO is the result of collaboration between the Drexel-led DRC-HUBO team, KAIST, and Rainbow, Inc., based on our team's analysis of performance of the HUBO2+ on early versions of the DRC task descriptions [14]. Once the DRC-HUBO was provided by KAIST/Rainbow, Inc., the team's efforts became focused on the development of algorithms and software implementation. In this section, we describe the DRC-HUBO's hardware briefly; the major changes from HUBO2+ will be highlighted.

## 2.1 DRC-HUBO hardware overview

DRC-HUBO (depicted in Figure 1) is 1.47 m tall with a wingspan of 2.04 m, weighs 52 kg (including battery), and has 32 degrees of freedom. Relative to its base model, HUBO2+, it was given longer and stronger limbs with an aluminum skeleton and shell, resulting in a taller and heavier robot. Figure 2 shows a comparison of DRC-HUBO with other robot platforms fielded in the DRC trials, in terms of hardware platform size and weight. DRC-HUBO is shorter and lighter than its average competitor in the DRC, which weighs 96.2 kg at a height of 156.4 cm [2].

The DRC-HUBO upgrade introduced major changes in limb design to accommodate the events of the DRC trials. The chief goals were to increase the robot's workspace, dexterity, and power. Compared with HUBO2+, length of the leg has been extended by 15%, and the length of the arm has been extended by 58%. The legs remain at 6 degrees of freedom (DOF), but the arms have been increased to 7 DOF. Arm payload was increased to over 3.5 kg at full stretch.

Grasping capabilities were upgraded as well. Three fingers on each hand close together via one motor for power grasps. On the right hand, there is an additional trigger finger which moves independently from the other three fingers, allowing the robot to operate power tools. The fingers can support up to approximately 9kg. Each hand also has a *peg* opposite the palm/fingers side, which can be used as a point contact when the robot is in a

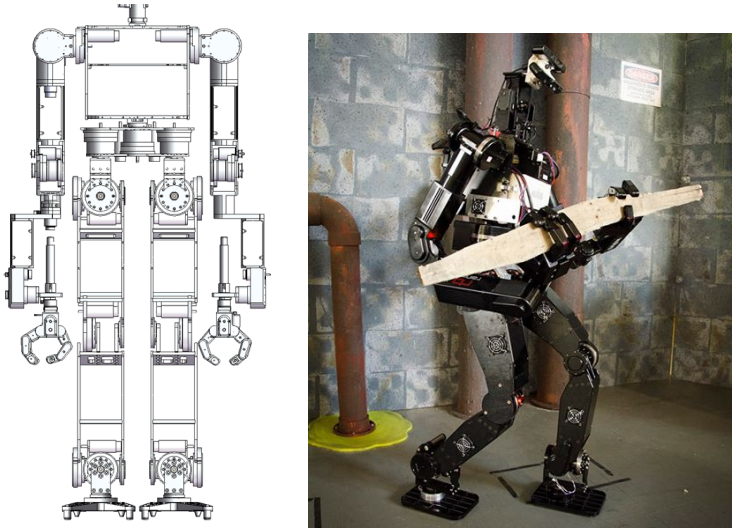

Figure 1: The DRC-HUBO humanoid robot. Left: design schematic. Right: in action (image courtesy of Drexel University).

quadrupedal walking mode (see Figure 3). The peg can also be used for other tasks such as turning the steering wheel and hooking the hose nozzle in the DRC tasks.

The joint motor driver modules were also upgraded, resulting in a size reduction of 18%. Additionally, two open-loop torque control modes (complementary switching and non-complementary switching) have been implemented. In non-complementary switching mode, the motor driver consumes less power and compensates back electromotive force (EMF) efficiently. This mode enabled us to implement a compliant controller for the arms (described in Section 3.2.2), and, thus, to make interaction between DRC-HUBO and the environment safer.

## 2.2 Perception system

The sensor head on the robot, shown in Figure 4, was designed and built by us. It pans $\pm 180^\circ$ and tilts $\pm 60^\circ$ without self-collision, and has the following sensors which are relevant to this work:

- **3 $\times$ Pt. Grey Flea3 cameras**, each with a $1280 \times 1024$ resolution and approximately $90^\circ \times 70^\circ$ field of view (FOV), forming a synchronized stereo rig with baselines of 6 cm, 12 cm, and 18 cm.

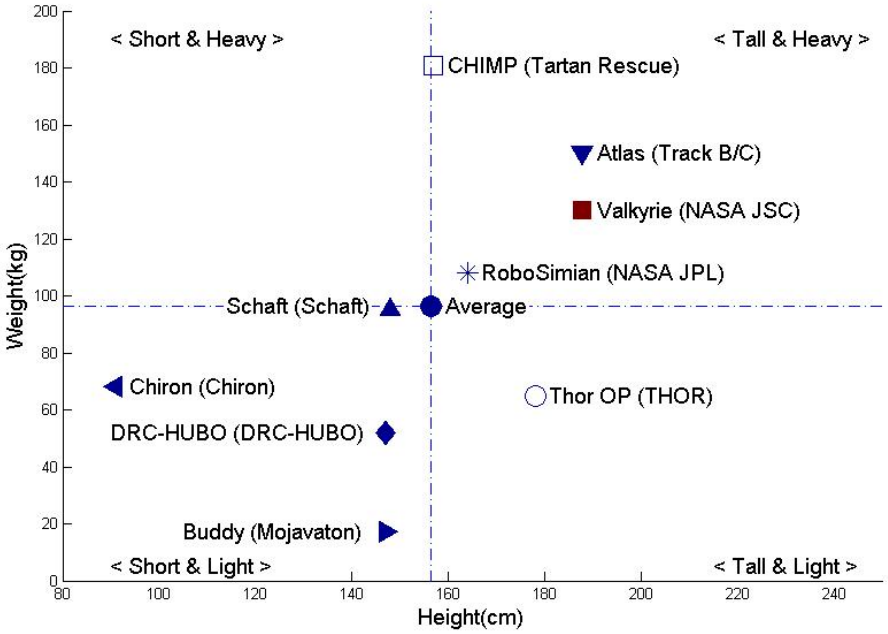


Figure 2: Comparison of robot platforms in the DRC trials.

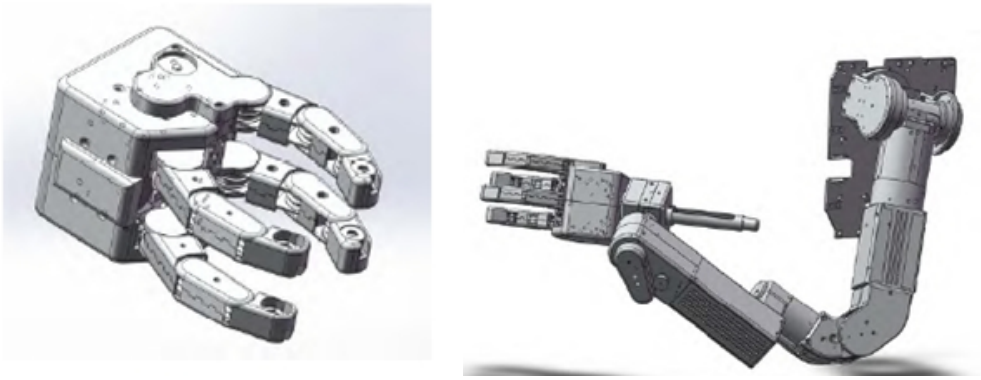

Figure 3: DRC-HUBO: hand and arm design. Left: robot's right hand with trigger finger. Right: arm with peg.

- **Hokuyo UTM-30LX-EW laser range-finder** (lidar) which scans at 40 Hz over a $270°$ FOV at an angular resolution of $0.25°$. The minimum detectable depth is 0.1 m

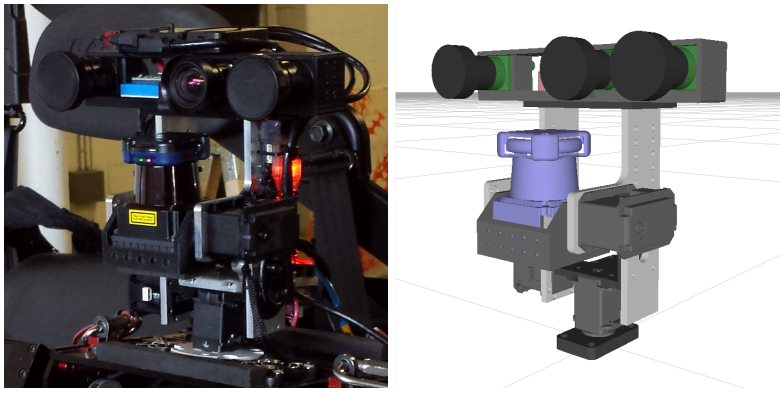

Figure 4: Sensor head photo detail and CAD model in ROS RViz [33].

and the maximum is 30 m, and intensity-like reflectance information is provided for each point. The Hokuyo is mounted on a dedicated tilting servo which has a range of $\pm 60^\circ$ for point cloud capture

- **Microstrain 3DM-GX3-45 IMU** with 3-axis accelerometer, 3-axis gyro, and GPS receiver/antenna

- **PrimeSense short-range RGB-D camera** (rear-facing) which captures RGB and depth images at $640 \times 480$ resolution with an FOV of $57.5^\circ \times 45^\circ$. The PrimeSense has a minimum range of 0.35 m and maximum depth range of 1.4 m, but it does not work in full sunlight.

The robot head geometry and sensors were chosen to cover the anticipated perception demands of the eight DRC events, based on the task descriptions promulgated by DARPA in the months before the competition, in a timely and robust fashion. These descriptions were changed multiple times, often significantly, and only became final in the last weeks before the competition. This uncertainty, and an early goal of complete autonomy rather than teleoperation, led to the creation of a head with somewhat more general capabilities than was ultimately required.

Perceptual tasks common to all of the events were grouped in several ways in order to assess sensor feasibility and placement, including working distance, required update rate, and minimum resolution. In the first category, for example, we broke tasks into: (a) *long-distance* ($> 5$ m away), including landmark, object, and obstacle detection for purposes of setting walking and driving navigation goals; (b) *mid-distance* (1-5 m away), including detailed terrain characterization for imminent obstacle avoidance and footstep planning;

and (c) *near-distance* ($< 1$ m), including object characterization and pose measurement for planning and monitoring grasping motions.

The sensors above were selected because they could collectively provide high-resolution 3-D and appearance information from the robot's toes or right in front of its face, out to tens of meters, in light, shadow, or darkness. Images were available at high frame-rate, coarse depth several times per second, and fine depth once every several seconds; and there was enough redundancy/overlap to complete the tasks even with one or two sensor failures.[1]

# 3 Software

The overarching software design focused on a distributed multi-process architecture. The primary goal behind distributing functional components into separate processes is to provide robustness and modularity. Robustness is achieved by designing processes to gracefully handle interruption in communication or operation. In the event of an unexpected crash, hold up, deadlock, or loss of communication from some other component in the pipeline, all other components are designed to behave reasonably and bring the system to a safe state while recovery is attempted. This is in contrast to a monolithic process design, which would produce a system-wide critical failure in the event of an unexpected crash or hangup in any functional component. Modularity is achieved by being conscious of the inputs and outputs of each process, and allowing the inputs to be agnostic to their sources while the outputs are available to all subscribers. With this framework, individual processes can be readily modified or replaced without adversely affecting the overall pipeline. The multi-process design also lends itself well to the distribution of computational load between the robot's onboard computers and the operators' computers. More detail on the design philosophy and motivation, especially during the early stages of development, is available in [18].

The diagram in Figure 5 provides an overview of the key processes (each box representing a process) in the system and their interconnection. The dotted line between the two major sections indicates which portions of the system are running on the robot's onboard computer versus which are running on the operators' computers. All processes on the operator computers run on top of the ROS framework [33]. Conversely, the processes running onboard the robot do not have any ROS dependency.[2] The ROS processes natu-

[1] The ladder-climbing task was performed "backwards", and while the head could have been rotated to point backwards, it was convenient to mount a separate rear-facing camera (the PrimeSense) which could also serve as a back-up to the front-facing cameras.

[2] With the exception of a separate perception computer, not depicted in Figure 5, that uses ROS to transmit camera images, lidar point clouds, and head pan/tilt commands.

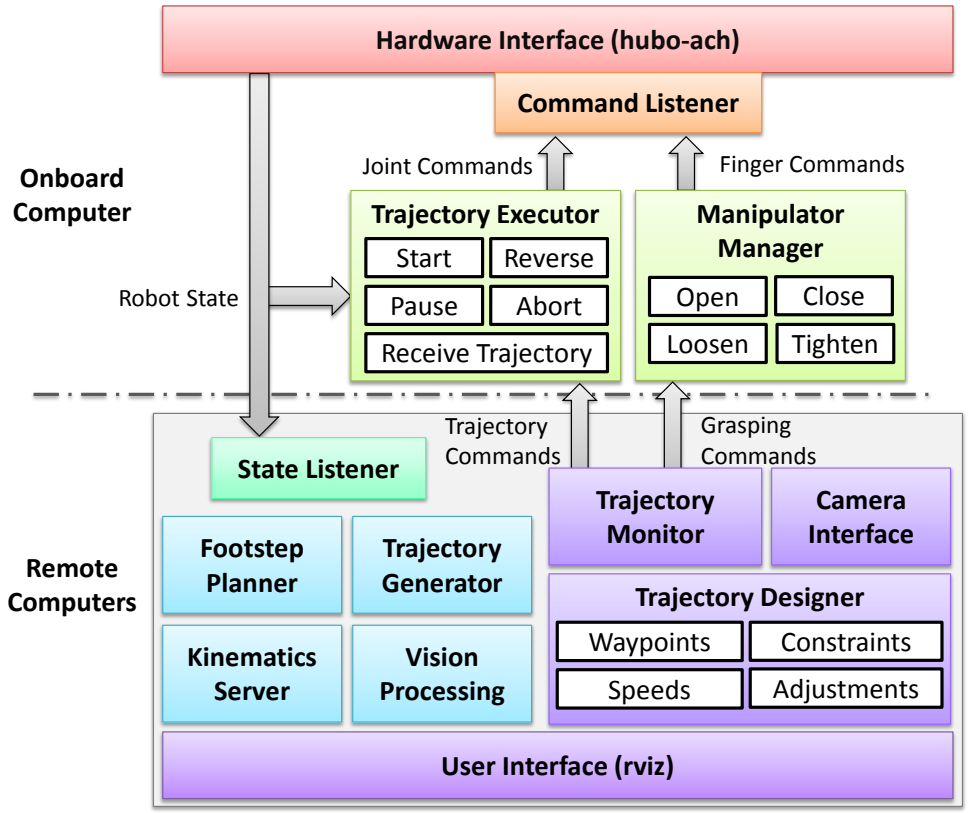


Figure 5: Diagram of our software architecture. Colored boxes indicate independent processes. These are color-coded (spanning the color spectrum) according to its location in the pipeline with red being hardware interface and purple being human interface. Gray arrows are Ach channels while the large gray box represents the ROS framework.

rally use a variety of ROS-based interprocess communication (IPC) methods such as topics and service calls. Onboard processes communicate via Ach channels [13]. Ach is an efficient IPC protocol designed specifically for real-time robotic applications. Some of its key features include serving processes with higher priorities before processes with lower priorities, priority inheritance (i.e. if a high-priority process is waiting on a lower-priority process, then the lower-priority process will temporarily inherit the higher priority until it is no longer blocking), and eliminating head-of-line blocking to ensure that a process is always operating on the latest available data.

The processes onboard the robot are capable of withstanding intermittent communica-

tion, and run asynchronously from the processes on the operator computers. Their general purpose is to perform real-time control on top of the whole-body trajectories provided by the operators. The processes on the operator computers are responsible for aiding the operators in constructing trajectories, providing the operators with situational awareness, and sending commands to the robot. These processes do not have real-time constraints and they do not need to be conservative with their computational resources, because the performance available for the operator workstations is not heavily constrained.

## 3.1 Low-level software

At the lowest level of software is the interface to the hardware. The software communicates with the hardware through Controller Area Network (CAN), specifically socketcan [24]. The control frequency of the robot is 200 Hz, constrained chiefly by CAN bus bandwidth limitations. In order to help maintain this frequency, Ubuntu 12.04 Server with a Preempt-RT patch was used, and processes were assigned priorities for the kernel according to their importance.

The trajectory executor serves as the second lowest level in the software hierarchy. It is responsible for receiving trajectories from the operator, and then executing these trajectories while performing real-time closed-loop control, which is discussed in Section 3.2. All trajectories which are sent to the trajectory executor begin and end in statically stable states. The entire trajectory must be received before execution begins. After the trajectory executor has completed a trajectory, it will continue to apply its real-time balance and manipulation control to the last configuration of the trajectory until the next trajectory is provided. When a novel trajectory arrives, the trajectory executor performs a "sanity check" to make sure that the first configuration of the new trajectory is identical to the last configuration of the previous one. This prevents catastrophic behavior that might result from old data packets being handled as if they are new, or if a misinformed trajectory gets sent.

The manipulator manager is responsible for the state of the robot's fingers, and is able to operate independently of the trajectory executor. Finger commands can be encoded into trajectories; additionally, the operator is able to override the trajectory's finger commands at any time through the manipulation manager. This feature is important in the event that a manipulation task does not go as expected, e.g. if the fingers need re-shaping before grasping is viable.

## 3.2 Real-time feedback controllers

Our system ran two active feedback controllers online: balance control and compliance control. To simplify the problem of applying a controller to a high-DOF humanoid robot, the upper body and lower body were each controlled independently. The degrees of freedom in the lower body were used to achieve balance by shifting the robot's pelvis, while the degrees of freedom in the upper body were used to achieve compliance through DRC-HUBO's upgraded non-complementary switching mode. The key objective of the balance controller was to correct for model error or unpredicted external forces during quasi-static motion and manipulation, whereas the objectives of the compliance controller were to relieve the strain of closed kinematic chains caused by interacting with the environment, and to lessen the severity of impacts between the robot and the environment.

### 3.2.1 Balance control

In each of the robot's ankles, there is a three axis force/torque sensor. Two axes read torque (these are along the longitudinal and lateral axes), and the third axis reads force along the vertical axis. Even though this means that two axes of force and one axis of torque are excluded from our available data, the three axes provided are enough to compute the zero moment point (ZMP), given information about foot angular orientation (we typically assumed flat ground). Computing the actual ZMP and comparing it to a desired ZMP provides an error vector which indicates the direction for the robot's pelvis to move such that the two can converge.

Our impedance controller is defined by the update rule

$$\ddot{\mathbf{s}} = -\frac{k}{m}\mathbf{s} - \frac{b}{m}\dot{\mathbf{s}} + \frac{1}{m}\mathbf{e}$$

Here, $\mathbf{s} = (x, y)$ is the the displacement vector which is applied to the pelvis location in order to adjust the ZMP, $\mathbf{e} = (\Delta x, \Delta y)$ is the error between the desired and actual ZMP location, and the $m$, $k$, and $b$ gains define the mass, stiffness, and damping of a virtual mass-spring system. In practice, we compute the time-varying value of $\mathbf{s}$ via Euler integration.

Since $\mathbf{s}$ and $\mathbf{e}$ are defined in workspace coordinates, it is necessary to use inverse kinematics (discussed in Section 3.3) to find the joint values corresponding to the desired pelvis displacement. Once these joint values are obtained, they are fed through to the motor control boards, which use relatively stiff PD control to reach their desired positions.

It is important to note that the balance control scheme is defined relative to the current waypoint along a trajectory. In order to prevent a sudden jump at the pelvis, the integrator state $(\mathbf{s}, \dot{\mathbf{s}})$ is carried over from one waypoint to the next. Hence, the integrator in

the controller serves both to eliminate steady-state error, and to enforce continuity of the output.

### 3.2.2 Compliance control

In order to perform compliance control, KAIST provided the DRC-HUBO with a new form of control in the firmware as mentioned in Section 2.1 called the non-complementary switching mode. This mode makes it possible to directly command Pulse Width Modulation (PWM) commands to the robot's joints instead of just position commands. Unfortunately, PWM does not directly map to torque, but we were able to construct a rough empirical relationship between the two and thereby do a pseudo-torque control of the robot. Two examples of these relationships can be seen in Figure 6.

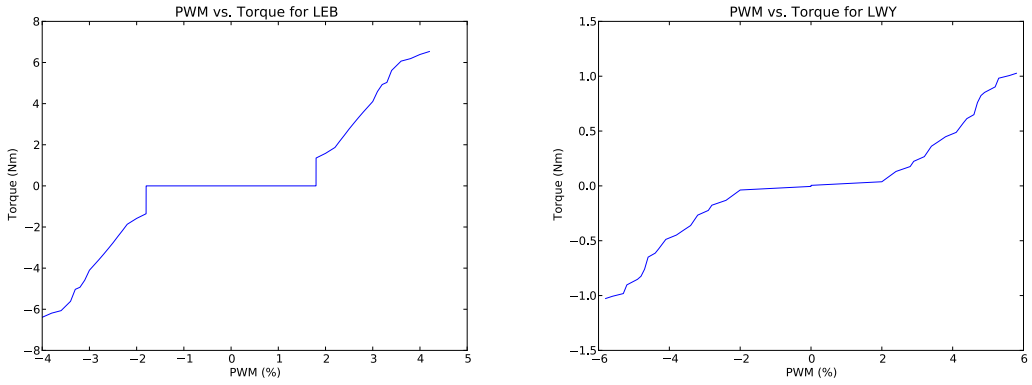


Figure 6: Empirical relationship between torque and PWM for two joints: Left Elbow (LEB) and Left Wrist Yaw (LWY).

In the plots, it is clear that there exists a deadband near zero PWM for some joints, where a non-zero command results in approximately zero torque output. This can be mainly attributed to friction in the joint driving system, and requires special compensation to overcome. Friction compensation was handled by giving a small boost in the direction of the joint's velocity as read by the encoder. This boost is proportional to the joint velocity, but capped to never go beyond the deadband. This gives a mapping from torque to PWM:

$$V_i = T_i^{-1}(\tau_i) + \min(K_{fi}\dot{q}_i, \text{sgn}(\dot{q}_i)V_{fmax,i}) \quad (1)$$

$$\tau_i = K_{pi}(q_{di} - q_i) - K_{di}\dot{q}_i + \tau_{Gi}(\mathbf{q}) \quad (2)$$

Here, $V_i$ is used to refer to PWM, and $T_i(V)$ is the torque of joint $i$ generated by a PWM of $V$. $T_i(V)$ is determined by the previously mentioned empirical mapping (Figure 6). Therefore $T_i^{-1}(\tau)$ is the PWM needed to achieve a torque of $\tau$. $K_{fi}$ is the proportional constant applied to the joint velocity $\dot{q}_i$ to overcome friction. $V_{fmax,i}$ is half the size of the deadband for joint $i$.

Using joint-wise PWM commands as the output signal, a combination of low-PD gain feedback control, feedforward gravity compensation, and friction compensation was used to achieve reasonably precise compliance control of the arms. The full law for the compliance controller is the combination of Equation 1 and Equation 2. In Equation 2, $q_{di}$ is the desired value for joint $i$, $q_i$ is the actual value of that joint, and $\tau_{Gi}(\mathbf{q})$ is the torque due to gravity for that joint given the robot's current configuration. In practice, $K_{di}$ has a value of zero because the internal damping of the motor is sufficient to prevent oscillations, but it could be used as a damping term if necessary.

There were no end effector force-torque readings being used in a feedback control law for the compliance control, because the wrists (in the same way as the ankles) only had three axes of force-torque readings available, which is not enough for a completely meaningful 6-DOF feedback controller. Even without force-torque sensor feedback, our control scheme was critical for handling closed kinematic chains and easing the challenge of manipulating objects in tight corners.

## 3.3 Kinematics and trajectory generation

Our trajectory generation software for the robot is based on efficient constrained interpolation between key poses. The key poses are provided via operator input, and are guaranteed to be both statically stable and free of self-collisions. Underpinning the trajectory generation software is a stable and efficient analytic inverse kinematics (IK) solver for the DRC-HUBO robot, along with a redundant state representation that allows for straightforward interpolation with end effector and body pose constraints.

### 3.3.1 Kinematics and limb IK

The DRC-HUBO robot has kinematically redundant arms, each with seven degrees of freedom, comprised by a spherical shoulder joint, an elbow, and a spherical wrist. Each leg has six degrees of freedom, comprised by a spherical hip joint, a knee, and ankle pitch and roll joints.

In the months leading up to the DRC, our subteam came to rely upon a fast, analytic inverse kinematics solver for both the arms and legs. Leg IK differs from the previous HUBO2+ robot only in terms of kinematic constants such as link lengths and joint limits;

however, due to the addition of a redundant degree of freedom, the arm IK was developed for the DRC, which we detail in Appendix A.

The IK solver for the arms assumes a fixed wrist roll angle (the final joint in the series leading from shoulder to end effector), and solves for the remaining six degrees of freedom. In our operator software, we effectively use the wrist roll parameter as an index into the kinematic null space of the end effector, allowing the operator to choose different arm configurations to either minimize torque on certain joints or avoid collisions, as needed.

### 3.3.2 State representation

Our trajectory generation system uses a highly redundant representation of robot state. In addition to the 27 joint angles for arms, legs, and waist, we also store a rigid transform and control mode for each end effector. The overall state of the robot is given by

$$\mathbf{q} = (\theta, \mathbf{T}_P, \mathbf{T}_{LF}, \mathbf{T}_{RF}, \mathbf{T}_{LH}, \mathbf{T}_{RH}, M_{LF}, M_{RF}, M_{LH}, M_{RH})$$

where $\theta$ is the vector of joint angles, $\mathbf{T}_{EE}$ is a rigid body transformation representing the position and orientation of an end effector $EE$, $M_{EE}$ is a control mode (described below), and $LF, RF, LH, RH$ index the end effectors: left and right feet and hands of the robot, respectively. $\mathbf{T}_P$ refers to the world-frame transformation of the pelvis of the robot, which is the root of the kinematic tree in our robot model. Each control mode $M_{EE} \in \{joint, body, world\}$ indicates whether the selected limb is currently being controlled at a joint level, or whether it is driven to its respective transformation via inverse kinematics.

We say a state is *resolved* when the inverse kinematics solver is invoked to modify the joint angles $\theta$ to correspond to the transformation $\mathbf{T}_{EE}$ of each end effector as needed, given the pose $\mathbf{T}_P$ of the root of the kinematic tree. When the state is resolved, if an end effector is in $joint$ mode, its joint angles are left unchanged. If it is in $body$ mode, then the IK solver is used to bring the end effector to the pose $\mathbf{T}_{EE}$ in body frame. Finally, if $M_{EE}$ is in $world$ mode, the desired effector pose in the body frame is given by $\mathbf{T}_P^{-1}\mathbf{T}_{EE}$ and the joints for that limb are determined accordingly by the IK solver.

Although the IK solver is analytic and not numerical, the angles contained in $\theta$ are nonetheless considered when resolving a state. For both the arms and legs, there are up to eight valid solutions for joint angles for a given end effector pose; our policy is to choose the one closest to the angles in $\theta$. When computing IK for the arms, we take care to consider the effects of the wrist roll joints as well as the waist joint. Although the IK solver does not ever affect these joints, their influence on the transformation between the pelvis and the hand must be accounted for. Accordingly, since these three joints are never modified by the IK solver, they are only changed due to operator input or interpolation (see Section 3.3.4).

### 3.3.3 Whole-body and center of mass IK

With the redundant state representation described above, performing whole-body IK (i.e. *resolving* a state) is a straightforward operation. When resolving a state, IK is performed independently for each limb relative to the pelvis, and a flag indicates whether the operation was successful. center of mass (COM) IK is also straightforward to perform. Given a desired position $\mathbf{x}_d$ of the robot center of mass, we can compute a desired displacement for the body (i.e. a translation to compose with $\mathbf{T}_P$) via the update rule

$$\mathbf{T}_P \leftarrow \left[\begin{array}{c|c} \mathbf{I} & \alpha\big(\mathbf{x}_d - \mathbf{x}(\mathbf{q})\big) \\ \hline 0 & 1 \end{array}\right] \mathbf{T}_P$$

where $\mathbf{x}(\mathbf{q})$ denotes the computed center of mass given the current state, and $\alpha$ is a step size (we find $\alpha \approx 0.5$ to give very fast convergence).

We note that this approach for COM IK does not compute whole-body Jacobians of the robot. Although we implemented code to compute the whole-body COM Jacobian given the end effector constraints, we found that in practice the COM Jacobian with respect to pelvis translation is close enough to a scaled identity matrix as to make the full computation much slower than the naïve update rule given above with no significant gain in accuracy. In benchmarks averaged across 300 regularly spaced robot positions, the naïve method outperformed steepest descent by over a factor of three (1.7 ms vs 3.4 ms per query, on average).

The chief disadvantage of implementing COM IK using the method outlined here is that it fails to exploit all degrees of freedom of the robot, especially the waist and unused limbs (i.e. moving a free arm to displace the COM, or rotating the torso about the pelvis to balance). Additional refinements allow us to operate in a mode where we only modify $\mathbf{T}_P$ if the computed COM lies outside of a conservatively shrunken support polygon of the robot.

### 3.3.4 Interpolation and trajectory validation

As mentioned earlier, trajectories (aside from walking) in our system are generated via smooth interpolation between two statically stable poses of the robot which are free of self-collisions. The algorithm for interpolation is relatively straightforward. Given two states $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, we begin by modifying the control modes and transformations in $\mathbf{q}_{init}$ to match the frames given in $\mathbf{q}_{goal}$; that is, end effectors are interpolated in whichever space is specified by $\mathbf{q}_{goal}$, be it joint angles, body frame, or world frame.

Next, we determine the duration of the interpolated trajectory by examining the maximum duration over all joint angles and poses, given acceleration and velocity bounds in each respective space. All interpolations are performed using cubic Hermite splines (see

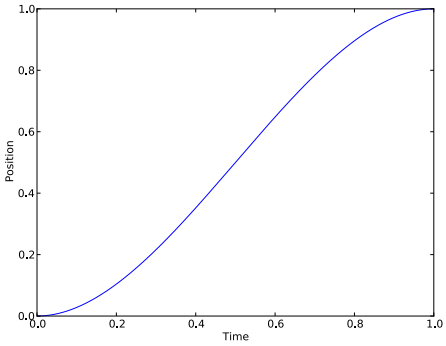


Figure 7: The cubic Hermite spline, defined by $x(t) = 3t^2 - 2t^3$. By scaling and shifting the axes, it can perform interpolation between arbitrary positions over an arbitrary duration, with zero velocity at the endpoints.

Figure 7). Such interpolation is straightforward for joint angles and the translation elements of rigid body transformations; rotations in 3D are interpolated by passing the output of a cubic Hermite spline as an input to the spherical linear interpolation function on unit quaternions. After determining the duration, we discretize time based on the control period of the robot, $\Delta t = 5$ ms. We then generate each interpolated state first by interpolating joint angles, second by interpolating transformations, and finally, by *resolving* the state as described in the previous subsection. The overall algorithm is listed in Algorithm 1.

The two different forms of interpolation prior to resolving IK serve two purposes. Interpolating joint angles maintains continuity in angles which are not modified by IK (such as wrist roll and waist rotation, and those of limbs in *joint* mode), and discourages "jump" discontinuities where IK swaps between valid solutions in configuration space. Interpolating transformations guarantees that the end effectors (and the pelvis) all move smoothly in the workspace (or body frame of the robot, as applicable).

Trajectories may fail to be valid for one of three reasons: the IK solver failed to find a solution, IK solutions for successive states are discontinuous in joint space due to joint limit violations, or the robot is found to be in self collision. The former failure is indicated by a failed call to the *resolve* function; the latter two are addressed by a validation function which checks for both failures.

For collision checking, we represent the robot as a union of simple convex objects such as capsules and boxes, as illustrated in Figure 8. For safety, the collision volumes are enlarged slightly compared to the actual physical dimensions of the robot. The *libccd* library [16] is used for collision detection. Collision detection for the simplified robot

**Data**: Resolved robot states $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$; velocity and acceleration limits $v_j, a_j$ for each joint $j$; rotational and translational velocity and acceleration limits $\mathbf{v}_{EE}, \mathbf{a}_{EE}$ for each end effector $EE$, and $\mathbf{v}_P, \mathbf{a}_P$ for pelvis

**Result**: A smooth trajectory connecting $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, with interpolation modes determined by $M_{EE}$ in $\mathbf{q}_{goal}$; or notification of collision or IK failure

**begin**

$t_{max} \leftarrow 0$

**for** each joint *j* **do**

$t_{max} \leftarrow \max\left(t_{max},\ duration(\theta_{init,j}, \theta_{goal,j}, v_j, a_j)\right)$

**end**

**for** each end effector *EE* **do**

place $\mathbf{T}_{init,EE}$ in the same frame as $\mathbf{T}_{goal,EE}$ as indicated by $M_{goal,EE}$

$t_{max} \leftarrow \max\left(t_{max},\ duration(\mathbf{T}_{init,EE}, \mathbf{T}_{goal,EE}, \mathbf{v}_{EE}, \mathbf{a}_{EE})\right)$

**end**

$t_{max} \leftarrow \max\left(t_{max},\ duration(\mathbf{T}_{init,P}, \mathbf{T}_{goal,P}, \mathbf{v}_P, \mathbf{a}_P)\right)$

$n \leftarrow \lceil t_{max}/\Delta t \rceil$

$\mathbf{q}_0 \leftarrow \mathbf{q}_{init}$

$\mathbf{q}_n \leftarrow \mathbf{q}_{goal}$

**for** $i = 1$ **to** $n - 1$ **do** `// assemble` $\mathbf{q}_i$

$u \leftarrow i/n$

**for** each joint *j* **do** $\theta_{i,j} \leftarrow smoothstep(\theta_{init,j}, \theta_{goal,j}, u)$ **for** each end effector *EE* **do** $\mathbf{T}_{i,EE} \leftarrow smoothstep(\mathbf{T}_{init,EE}, \mathbf{T}_{goal,EE}, u)$

$\mathbf{T}_{i,P} \leftarrow smoothstep(\mathbf{T}_{init,P}, \mathbf{T}_{goal,P}, u)$

**if not** $resolve(\mathbf{q}_i)$ **or not** $validate(\mathbf{q}_{i-1}, \mathbf{q}_i)$ **then return** $failure$

**end**

**if not** $validate(\mathbf{q}_{n-1}, \mathbf{q}_n)$ **then return** $failure$ **return** $success$ with trajectory $\mathbf{q}_0 \ldots \mathbf{q}_n$

**end**

**Algorithm 1**: Smooth interpolation with end effector and pose constraints. The *duration* function computes the minimum duration of a cubic Hermite spline given velocity and acceleration bounds, *smoothstep* performs cubic Hermite spline interpolation on angles or rigid body transformations, *resolve* performs whole-body IK, and *validate* checks for continuity and self-collisions.

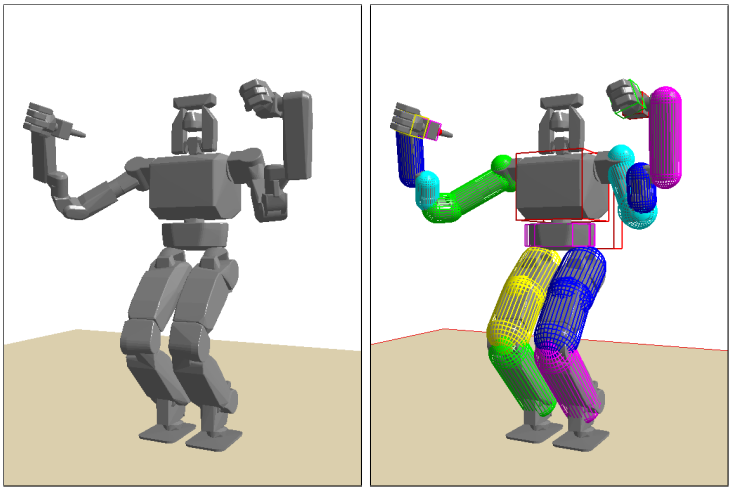

Figure 8: Collision model of DRC-HUBO. Left: the triangle mesh model, generated from CAD. Right: we conservatively approximate the robot as a collection of simple convex objects such as boxes and capsules, allowing very efficient detection of self-collisions.

geometry is fast when compared to a triangle mesh representation. In practice, computing a naïve $O(n^2)$ collision check on all $n$ links of the robot with our simplified collision representation is almost four times faster than triangle mesh collision checking with the state-of-the art FCL library, which uses sophisticated spatial data structures to gain high performance [28]. In an experiment averaged across 20,000 randomly generated joint configurations, the simplified checker took 0.13 ms per query, whereas triangle mesh checking took 0.45 ms. Of the 20,000 randomly sampled configurations, 39% were found to be free of self-collisions using the triangle mesh checker, and only 21% using the convex approximation, reflecting that the latter conservatively overestimates the volume occupied by the robot.

Although our system does not guarantee that the interpolated trajectories are dynamically stable, in practice the online balancing controller (Section 3.2.1) was effective over quite a large range of acceleration and velocity profiles, and we observed very few problems with balance stemming from the motion of trajectories themselves, as opposed to forceful interactions with the environment.

## 3.4 Walking

For our system, we implemented a dynamically stable walking trajectory generator using a ZMP preview controller to generate whole-body trajectories [21]. Although we exper-

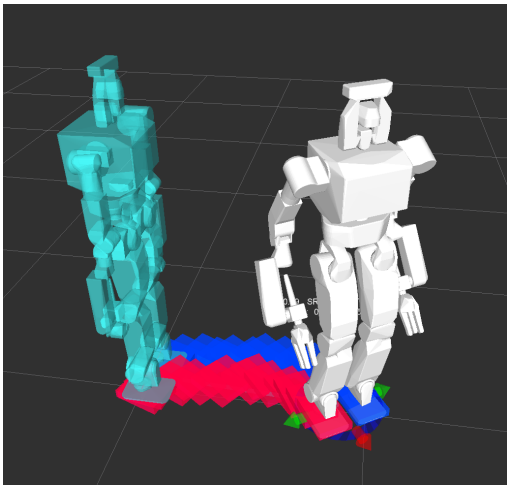

Figure 9: Our footstep planner automatically computes a sequence of steps to bring the robot to a desired destination. Current robot state is the translucent cyan model, planned state is white, and the interactive marker (arrows) allows the operator to translate and rotate the destination in the $(x, y)$ plane.

imented with many forms of online feedback control on top of the ZMP preview controller trajectories, we were unable to reliably outperform simple closed-loop PD control at the joint level. Although our admittedly simple walking controller worked reliably in controlled laboratory conditions, it was not as robust as desired in the actual conditions observed at the DRC trials (see Section 5.1.3 for details).

Operators could generate any of a number of regular walking gaits, including walking forward/backward, turning left/right, and sidestepping left/right. Additionally, since the ZMP preview controller generates trajectories based solely on footstep poses and timings, it was straightforward to produce a footstep planner [9, 10] capable of generating a sequence of footsteps to bring the robot to an arbitrary goal position and orientation (see Figure 9).

Both trajectories for walking and the interpolated trajectories defined in the previous subsection were represented both in memory and on the network as a sequence of joint configurations sampled at the control frequency of the robot, and augmented with additional metadata, including whether the compliance and balance controllers (Section 3.2.1 and Section 3.2.2) should be active, as well as the desired location of the COM or ZMP.

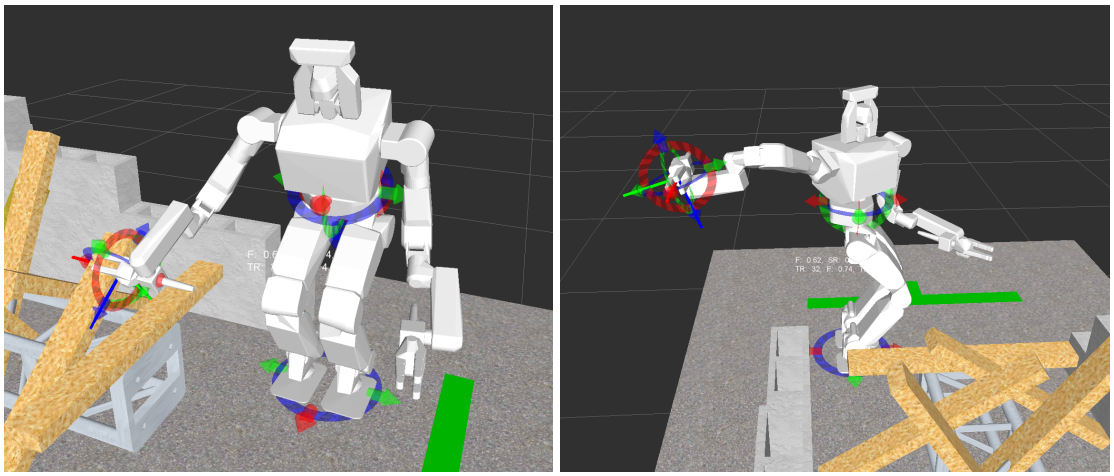

Figure 10: Sample key poses for the debris removal task. We created 3D models approximating each DRC event in order to build a library of key poses to stitch together into trajectories. Left: grasp pose for a diagonally oriented piece of debris. Right: pose to safely drop a wooden board. The hand is oriented such that the wood will slide down through the fingers before they are fully opened; furthermore, the board will be angled to fall away from the course when it hits the ground. These details are relatively easy for a human operator to provide, but difficult to encode in a general manner for autonomous systems.

## 3.5 Operator tools and communications

We divided the operator tasks into three main roles. The *trajectory designer* is responsible for construction and sequencing of key poses which are connected via constrained interpolation (see Section 3.3.4). The *execution manager* is responsible for sending trajectories to the robot and monitoring their execution in real-time. Finally, the *perception manager* is responsible for gathering images and point cloud data to enable the other two operators to perform their tasks. All operator tools are implemented on top of the ROS RViz program, whose plugin architecture makes it relatively straightforward to integrate robot models, 3D assets, and perception data such as images and point clouds.

### 3.5.1 Trajectory designer

Since the DRC events were specified with a relatively high degree of precision prior to the challenge, we developed a library of good candidate key poses for each task. For example, in the debris removal task, we were able to pre-select the general robot configurations for pregrasp, grasp, lift, and drop phases for each piece of debris. Similarly, for door opening, we could choreograph the approach to each door, again selecting poses for pregrasp, grasp, turning, pushing, etc. Example key poses are illustrated in Figure 10 and Figure 11.

Our RViz-based tools provide interactive markers which allow the trajectory designer to manipulate the robot's feet (which are always moved in tandem – in our system, their relative offset never changes except when the robot is walking), hands, and pelvis. Separate controls on a dockable panel provide functionality to adjust the waist angle and wrist roll angles (see Figure 12), as well as to modify end effector poses numerically. From the poses of the interactive markers, the full robot state can be inferred.

As a safety measure, the operator tools automatically use the COM IK procedure outlined in Section 3.3.3 to ensure static stability of every key pose, and they prevent the operator from generating any pose which places the robot into self-collision, or which violates joint limits. Hence, the start and end points for interpolation are guaranteed to be valid.

In addition to the pose of the robot, the trajectory designer selects for each destination key pose the desired interpolation mode (joint space, body-frame, or world-frame) as well as the speed and acceleration limits for joints and end effectors. Although our default choices (max end effector linear velocity of 10 cm/s, max end effector angular velocity and joint velocity of 0.8 rad/s, with nominal acceleration to reach max velocity over approximately 1 s) are reasonable for a large variety of motions, the operator may choose to slow down during complex manipulation procedures that require monitoring, or to assist the balance controller (Section 3.2.1) when manipulating heavy objects such as the power tool or doors.

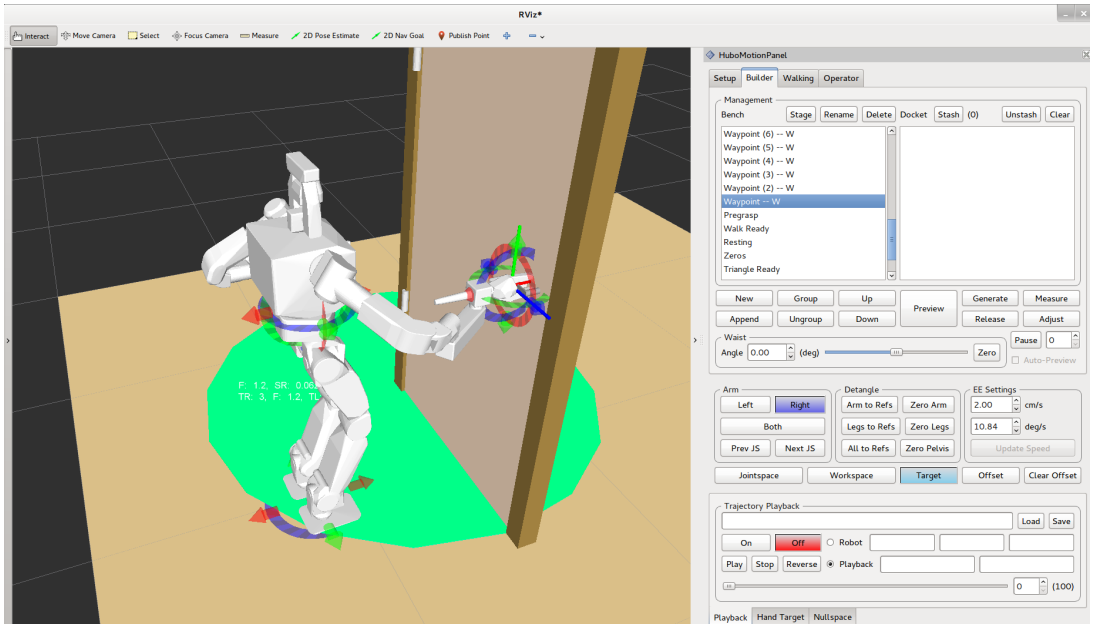


Figure 11: Sample key pose for the door opening task, along with a subset of our RViz user interface. The interactive markers (arrows on left hand side) allow the operator to position the robot's feet, pelvis, and hands.

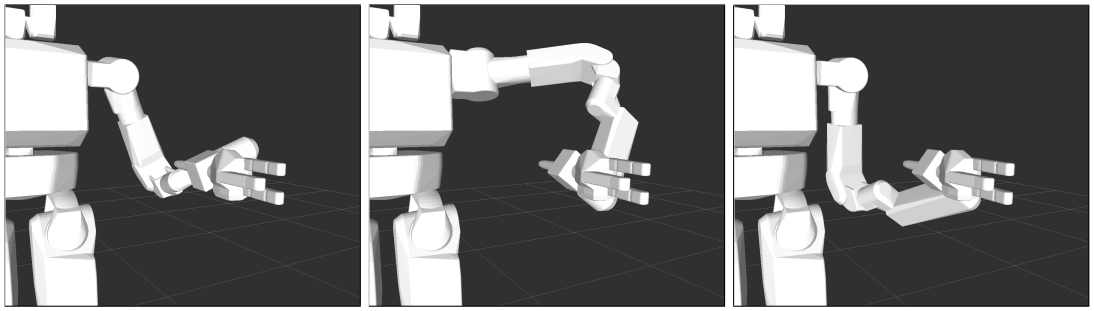

Figure 12: Moving the 7-DOF arm through its nullspace. Our system allowed operators to interactively explore the nullspace of the arm, either by incrementing the wrist roll angle (left two images), or by numerically minimizing the torque due to gravity at the shoulder yaw joint (right image), as it was the one most prone to overloading.

During operation, the trajectory designer is responsible for selecting an appropriate key pose from the library and modifying it to reflect the environment the robot is situated in (see Figure 13). For instance, if walking has delivered the robot to a slightly different position and/or orientation than desired, the designer may modify the end effector pose slightly to accurately grasp a target object. When the trajectory designer is satisfied with a current key pose or sequence of key poses, they are joined into a trajectory via interpolation and sent via a custom ROS message to the execution manager.

The trajectory designer can specify walking destinations for the robot by dragging the interactive foot marker. When the marker is dragged, the footstep planner (described in 3.4) automatically computes a sequence of footsteps to bring the robot from its current position to the relative pose specified by the marker (see Figure 9). Hence, a reasonable approach can be computed by placing the robot model into the key pose for grasp or pregrasp and dragging it to the desired position in the current point cloud scan.

### 3.5.2 Execution manager

The execution manager is responsible for sending trajectories to the robot and monitoring their execution. In the case of an unanticipated event such as a collision with an obstacle or a failed grasp, the execution manager pauses trajectory execution and determines an appropriate course of action. Minor errors may be corrected by refining the key pose specifying the endpoint of the trajectory, and are easy to correct by small manipulations of interactive markers; whereas major errors will likely require attention from the trajectory designer. The execution manager is also responsible for taking small footsteps to correct errors in approach if the walking controller has not arrived precisely at its destination due

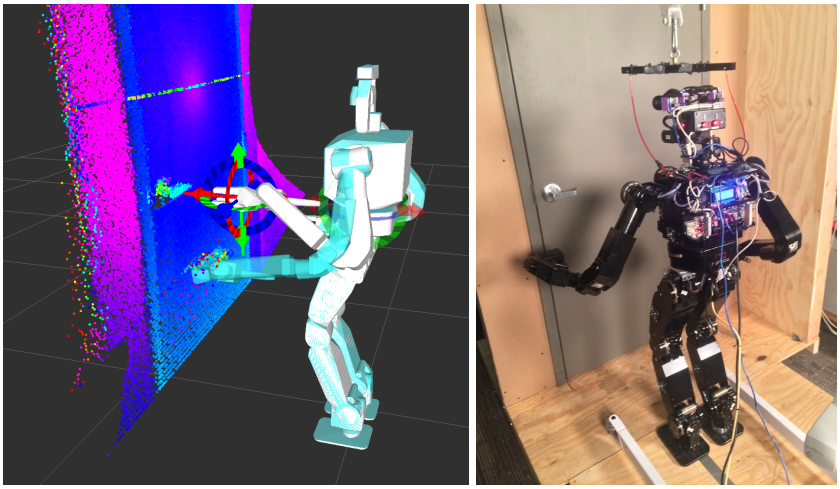

Figure 13: Left: screenshot of an operator's view in RViz during the door opening task. The transparent cyan robot model is the actual robot state, the solid white model is the operator's puppet, and the point cloud is a lidar scan of the door. Right: the actual robot and door.

to accumulated errors in odometry. For these small adjustments, either the footstep planner or the regular gaits may be used.

Another responsibility for the execution manager is determining which controllers should be active during each trajectory execution. The balance controller is typically enabled at all times, except when pushing or pulling heavy objects in the environment. During such times, the balance controller is prone to quickly build up internal forces due to positive feedback. We used compliant control of the arms extensively for the door opening task as well as the wall breaking task; for debris removal, since the manipulated objects were relatively light, we privileged manipulation accuracy over compliance and ran the arms using the traditional stiff PD controller.

Finally, the execution manager is responsible for operating the robot's hands. The DRC-HUBO has three degrees of freedom on the fingers, a power grasp for each hand, as well as an independently controllable "trigger finger" on the right hand which enables the operation of power tools. The fingers are velocity controlled, and hence the operator can choose whether to open them, close them, or leave them in a relaxed state which generally preserves their shape but exerts no force. Since there is no force or torque sensing in the fingers (only a 3-DOF force/torque sensor at the wrist), feedback for grasping is visual, mediated by the camera and the lidar.

Although some responsibilities such as walking and trajectory refinement are shared by the trajectory designer and the execution manager, we tried to maximize their productivity

by engaging them in parallel as much as possible. For example, as the execution manager is monitoring the current trajectory, the trajectory designer can be preparing the next one.

### 3.5.3 Perception Manager

The perception manager is responsible for gathering the necessary data to allow the other two operators to perform their jobs well. The sensor head (described in Section 2.2) is driven by three Dynamixel servo motors (two for overall pan-tilt and one for the lidar tilt), which are driven independently of the other joints of the robot. Accordingly, the head pose can be adjusted in real time by the perception manager.

Aside from aiming the head, the perception manager is also responsible for managing communications bandwidth for perception and modifying sensor parameters. During times when monitoring is critical, the frame rate of images from the camera may be increased, or it may be decreased when bandwidth is needed to upload trajectories to the robots. The transmitted image resolution, video quality, and region of interest for auto-exposure are also parameters which can be adjusted on the fly.

Unlike camera images which are streamed at an operator-determined rate, lidar point cloud data is collected from the robot only upon request from the perception manager. To further control bandwidth, each cloud can be filtered by defining horizontal and vertical angular limits, maximum range, and downsampled through voxelization before being sent back to the operator workstations. We found that as the operator with the most experience working with point cloud data, the perception manager was often helpful in fine-tuning the robot's hands while approaching very tight grasps such as the doorknob and drill handle.

### 3.5.4 Communications

All perception data, key poses, and trajectories are shared among operator computers via ROS messages. Robot state and proprioceptive sensor readings are sent to the execution manager via Achd (the network layer for Ach) over a compressed ssh connection, and throttled to $\sim$2 Hz to minimize bandwidth, and trajectories are sent to the robot via Achd as well.

To conserve bandwidth, compressed camera images and pointclouds (using JPG and zlib compression, respectively) are sent to the perception manager workstation from the robot via ROS. We used WPI's *teleop_toolkit* for image transport, which allows the operator to tune camera frame rate and image quality in real time [31].

# 4 Lab Tests and DRC Trials

The DRC trials, held from December 20-21 2013 at the Homestead-Miami speedway, constitute one of the three milestone competitions in the DRC: the Virtual Robotics Challenge (VRC), the DRC Trials, and the DRC Finals. At the trials, 16 teams competed on eight events that were designed to test both robot capabilities and effective operator control schemes. The teams participating at the DRC trials were comprised by six track A teams, seven track B/C teams, and three track D teams. The seven track B and C teams were selected through the VRC, which was conducted in June 2013 and which tested teams' abilities to effectively control a standardized robot model through three tasks in a simulated environment [2]. After the VRC, DARPA provided a physical Atlas robot, designed and manufactured by Boston Dynamics, Inc., for the track B and C teams. In the meantime, teams from track A (such as our own) and track D developed their own hardware platforms as well as software. Each of our three events in the DRC trials must be completed in a maximum of 30 minutes (minus a time penalty of five minutes per human intervention), with an additional 15 minutes of setup time provided beforehand.

In the remainder of this section, we describe the three DRC trial events targeted by our subteam[3], along with experimental results. The majority of the development and testing of our system took place at the Humanoid Robotics Lab at Georgia Institute of Technology (GA Tech), starting around August 2013. Although we did our best to simulate the DRC trials, the test conditions at GA Tech were under our control. To obtain a more objective evaluation of performance, team DRC-HUBO held a "dry-run" at Drexel University in mid-November, where each subteam demonstrated their respective systems in conditions which represented the DRC trial conditions and rules available at the time as accurately as possible.

## 4.1 Debris removal

The course layout for this event is shown in Figure 14. In this event, the robot is required to remove ten pieces of lightweight lumber (e.g. balsa wood) which are distributed along with a metal truss between two partial walls made of cinder blocks in front of a doorway, and subsequently walk through the door. Teams can a score maximum of four points, based upon completion of three sub-tasks: removing the first five pieces of debris, removing an additional five, and walking through the doorway. An additional point is awarded for completion of all three sub-tasks within the time limit without human intervention.

[3]Due to space considerations, some details and rules are omitted. For more complete descriptions and rules, we refer readers to [2, 15].

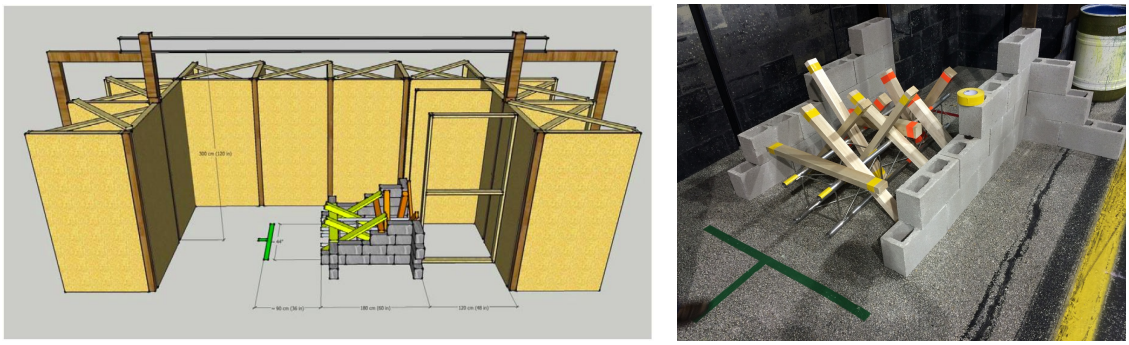

Figure 14: The debris removal event. Left: event schematic (from [15]). Right: actual setup at DRC.

### 4.1.1 Debris removal results

As DRC-HUBO hardware arrived in the lab in August, we started implementing algorithms on the hardware platform, and began regular lab tests with DRC-HUBO. Early on, we barely cleared a piece within the time limit because the teleoperation system was immature, and the operators also needed practice. Software development and practice went side by side. As the DRC trials approached, DRC-HUBO could clear five debris pieces within the time limit. Performance at the mid-November dry-run at Drexel also indicated that DRC-HUBO could score at least one point by removing the first five debris pieces. During our final practice for debris clearing on the morning of the actual trial, DRC-HUBO performed as expected by clearing five debris pieces within the time limit.

Changes in the event specifications prevented us from showcasing some capabilities we developed but never used. To cope with the initial requirements, which specified both larger and denser lumber, we adopted a dual-arm manipulation strategy (see Figure 15) that was straightforward to implement under our trajectory generation system. Subsequent modifications of the specifications obviated the need for dual-arm manipulation, which ended up being slower than single-arm, with a much smaller kinematic workspace.

### 4.1.2 Power failure

The debris removal task was our subteam's first event at the DRC trials. In order to fit into the 15-minute setup time, we performed a portion of our startup and calibration procedures off-site, and maintained system power using a uninterruptable power supply (UPS) while transporting the robot to the event. At the start of the debris removal task, the UPS failed, causing the robot to fall over. We were able to repair most damage to the robot and run

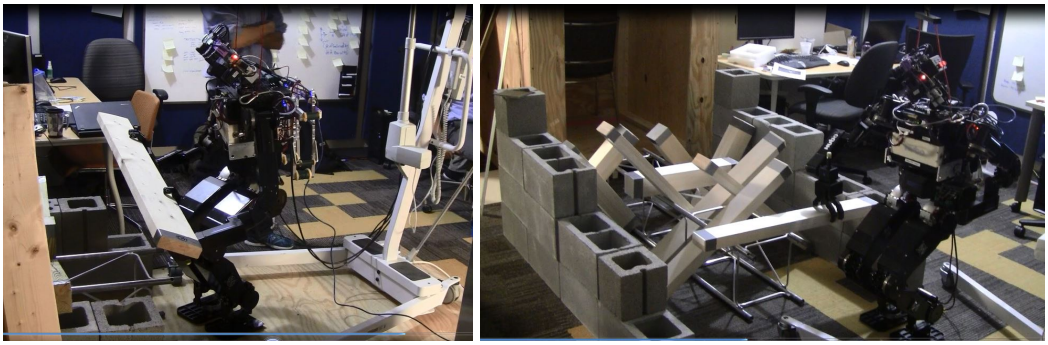

Figure 15: Debris removal testing at GA Tech. Two-armed grasping for heavier pieces (left) was implemented early on, but became unnecessary as the task descriptions evolved.

without the UPS for the rest of the trials, but the failure had several persistent negative consequences: we were forced to overrun our 15 minute setup time for each of the other two events, compressing the time available to complete them; the head was bent during the fall, resulting in poor alignment of camera images and lidar point clouds; and finally, several leg joints became slightly miscalibrated, impeding walking and balancing.

## 4.2 Door opening

For this event, the robot must open and pass through three doors installed on a flat floor. The first door must be pushed open by the robot, and the second two open via pulling. A weighted closer mechanism is installed on the third door. The robot earns a point for each door traversed, with a bonus point awarded for traversing all three without human intervention. See Figure 16 for an illustration of the task.

### 4.2.1 Door opening results

Photos in Figure 17 were taken during a small-scale mock-up test in the lab at GA Tech. In our lab, DRC-HUBO could reliably enter a push door, and, in some cases, enter a pull door within the time limit. We varied our strategies for door opening throughout testing and development. Typically, the robot pushed on the door with a forearm, either with the pelvis oriented sideways relative to the door, or forwards towards the door, in what we termed the "football stance". Once the door was opened slightly, we would depend upon the walking motion of the robot to naturally push the door open the rest of the way as the robot walked through. Walking speed was limited by stability concerns – applying too much force to the door would tend to overbalance the robot. Our strategy for pulling, on

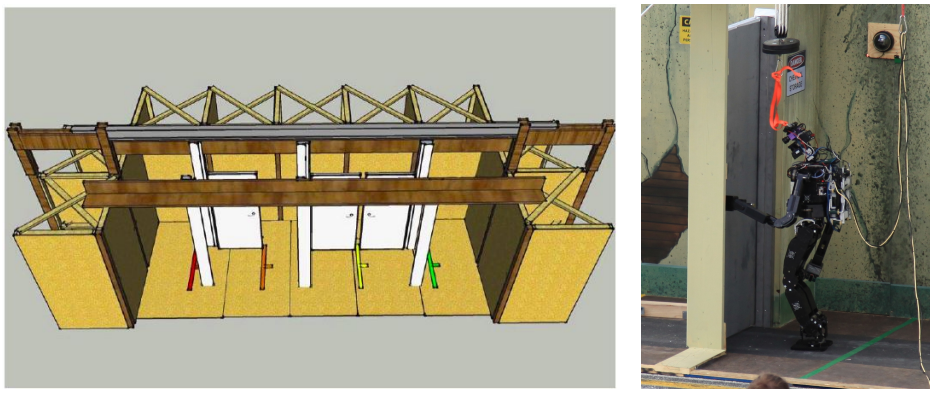

Figure 16: The door opening event. Left: event schematic (from [15]). Right: our system participating in the event at the DRC trials.

the other hand, was to situate the robot's feet outside of the arc of the door before pulling, so the robot could open it enough to walk through without manipulating the door while walking.

Due to space limitations, only one door was installed in our test mock-up at GA Tech. This required DRC-HUBO to turn around 180° to open the "pull door" after successfully walking through the "push door" (although both were, in fact, the same door). Even under this limitation, both routine lab tests and the dry-run rehearsal at Drexel showed that DRC-HUBO could score at least one point by completing the first sub-task. We believe our system's consistent performance–despite substantial differences between the mock-ups at Drexel and in our own lab–showcases the effectiveness of our general-purpose teleoperation system.

The door opening task was our subteam's second event in the actual DRC trials. Despite the robot's compromised state after the power failure, we were able to approach and open the first door; unfortunately, the strong wind (which we had neither encountered nor considered during testing, as all of our tests were conducted indoors) blew the door shut. After we opened the door once more, and as time was running out, we attempted to side step through the doorway much more quickly than we typically did during practice, and the robot lost balance and fell over, ending the trial.

## 4.3 Wall breaking

For this event, the robot was required to approach and pick up a cordless drill with a horizontal cutting bit, and subsequently use it to make several prescribed cuts in a nearby

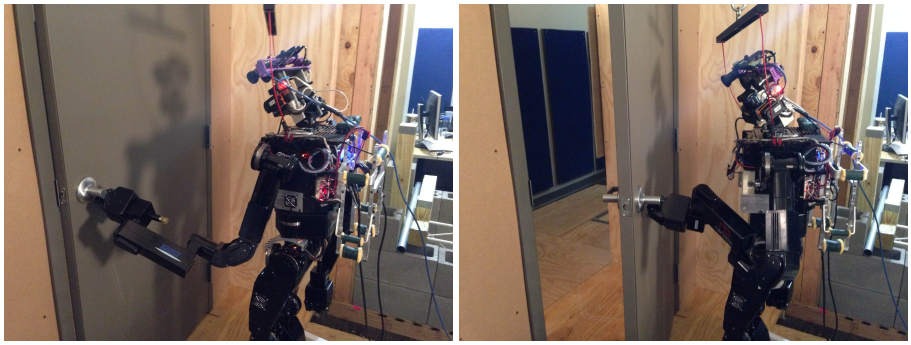

Figure 17: DRC-HUBO performing door opening at GA Tech.

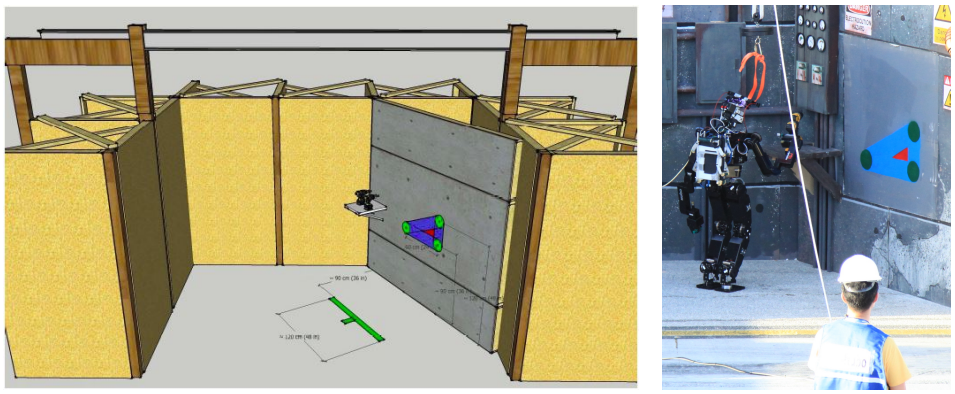

Figure 18: The wall breaking event. Left: event schematic (from [15]). Right: our system participating in the event at the DRC trials.

slab of drywall. Up to three points are awarded for successfully cutting each edge of a 2 foot $\times$ 1 foot right triangle, with a bonus point awarded for removing the entire triangle. The event is illustrated in Figure 18.

### 4.3.1 Wall breaking results

Among the three events describe in this chapter, we found wall breaking the most challenging, since the robot must to use a tool to interact with and eventually modify the environment. The task also involves locomotion while holding a heavy object (i.e. the drill). A key decision we made regarding the tool usage and interaction with the environment was to use one arm to operate the drill. This meant we did not fully exploit one of

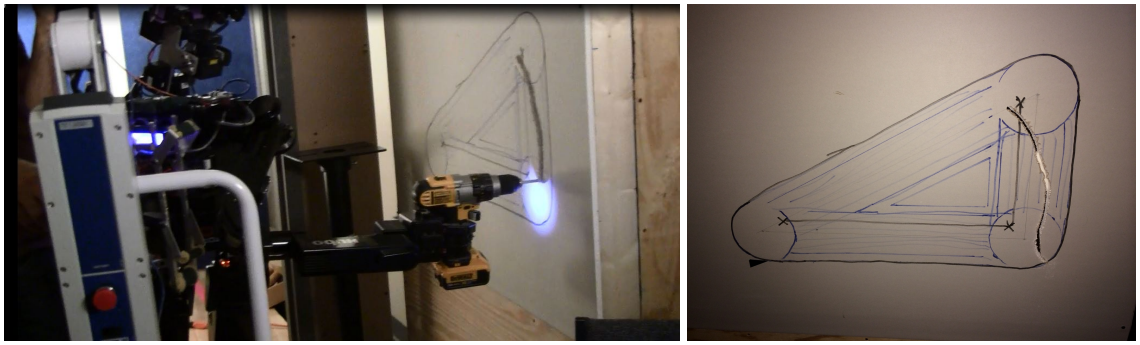

Figure 19: Wall breaking task test at GA Tech.

the main advantages of the DRC-HUBO design over HUBO2+. In fact, DRC-HUBO's arms were designed with dual-arm manipulation in mind, which initially seemed necessary for the wall breaking task. However, because (i) grasping the drill with two hands is a challenging task by itself, (ii) closed kinematic chains formed by dual-hand grasping and drilling on the wall may cause more frequent motor burn-outs, and (iii) dual-arm motion trajectories are far more highly constrained than single-arm, we decided to use one arm for the wall breaking task.

Grasping the drill was a very time-consuming operation for us, made especially difficult by the very narrow window for proper grasping. If the robot's hand is just slightly too high, too low, or rotated from a straight-on grip, the trigger can not be depressed successfully by the robot. Furthermore, drilling with one arm tended to overload the shoulder joints, especially shoulder yaw. Interaction between a spinning drill bit and the wall also applied non-negligible disturbance forces to DRC-HUBO, and made it difficult to cut a clean, straight line (as illustrated in Figure 19).

Our performance on this task varied. As DARPA refined the rules and task descriptions in the months leading up to the DRC trials, we adopted their changes to the event mock-up in our lab, and were forced to change our strategy accordingly. Ultimately, in both the lab at GA Tech and the dry-run at Drexel, DRC-HUBO could reliably cut one, and occasionally two edges of the triangle within the time limit, or before shoulder motors overheated. Based on our progress and the test results, we expected DRC-HUBO to score one point by cutting one edge in the actual DRC trials. This was somewhat corroborated during a separate, brief dry-run staged by the DRC organizers in Homestead, where our system successfully cut into the wall (albeit not along one of the prescribed lines).

In the actual DRC trials, the wall breaking task was our subteam's last event. In the trial run, our damaged DRC-HUBO managed to walk to the drill, but the first grasping

attempt was incomplete, and the robot fell over during the second grasping attempt from a different angle. After intervention, DRC-HUBO walked close to the drill but fell over again, ending the trial.

# 5 Lessons Learned

The DRC differs in both degree and in kind from most robotics research projects. Under many metrics (hours of robot time, lines of code, size of team), the DRC exceeds the typical project experiences of the authors, by far. Whereas a typical project might involve demonstrating a single research innovation in planning, control or perception on robotic hardware, the DRC ranges the entire spectrum of robotic systems, from mechanical and electrical design, to low-level device drivers, to high-level behavior generation. Since the conclusion of the DRC, we have had some time to reflect on the lessons that such a demanding challenge taught us.

## 5.1 What went wrong

### 5.1.1 Event-based task allocation

As discussed in Section 1, our team was highly distributed, with participation from nine U.S. institutions, and hardware support provided by KAIST and Rainbow, Inc. in South Korea. An early, high-level strategic decision was to allocate each subteam's efforts roughly by event, rather than by a systems-based approach. Consequently, there was significant duplication of effort among the entire team in many key areas. Many of the subteams independently arrived at their own solutions for functionality that was shared among various events, such as walking, user interfaces, constrained manipulation, perception, and communications, to name just a few examples.

Duplication of code on a project like the DRC has a number of drawbacks. Not only is duplicated functionality less well tested than common code, it is less likely to have been written by the most relevant expert. Since subteams' expertise varied with respect to planning, control, perception, and manipulation, and each subteam was developing their own functionality, few software systems took full advantage of the joint knowledge of the entire team.

A distributed, event-centric task allocation also creates perverse incentives in that time spent on releasing and maintaining shared code is time not spent on one's own event. For us, walking and dynamic balancing (see Section 5.1.3) was the most prominent, but by no means the only example. It is tempting to suggest that a systems-based or competency-based task allocation would have been more effective for our entire team; however, such an

approach has its own difficulties. Knowing *a priori* precisely which functionality should be shared across subteams presumes global knowledge about the top-level organization of the system before it has been designed or implemented. Faced with a future DRC-like project, we believe that some combination of the two approaches would probably be best.

### 5.1.2 Managing complexity

Aside from task allocation, an important question is how to mitigate the inherent complexity of robotic systems. Compared with more traditional software-only development, dependencies between functional units are complex, and mediated by interaction with the physical world through a diverse array of electromechanical systems.

For the DRC, with its broad scope and rapid development schedule, managing and understanding complexity was crucial. One illustrative example arose while debugging our walking controller, which exhibited some puzzling sporadic failures early in its development. After weeks of verifying kinematics and tweaking parameters in the walking controller to little avail, we finally discovered that the root cause was a timing glitch – only triggered occasionally – in the low-level program responsible for communicating with the motor control boards.

This speedbump taught us two lessons. The first is to endeavor to be as unbiased as possible when identifying root causes. Although every program in the system was potentially a source of error, we incorrectly focused on the novel component we were developing, rather than code which preexisted it. The second lesson is to test every subsystem to the greatest extent possible. Writing unit tests for a hardware-in-the-loop controller might be a difficult and time-consuming task, but it could have prevented this problem.

### 5.1.3 Walking and dynamic stability

When DARPA announced that the Atlas robot would be the government furnished equipment platform for the DRC, it became clear that balancing and walking would be fundamental to success. Although our team had implemented several walking and dynamic balancing controllers early on in the project, none was particularly robust, and this remained a persistent weak spot for our entire team.

Looking back, it would have been prudent to specify a minimum set of requirements for walking and balancing, including resistance to various types and sizes of perturbations, as early in the project as possible. Another lesson, related to the event-based task allocation, is that if no resources are explicitly allocated for common functionality, developers will tend to focus on the difficult aspects unique to their own subproblems – even if the missing common functionality entails a high risk of failure.

### 5.1.4 Sensing

Overall, the sensor head seemed adequate for carrying out the DRC tasks based on performance during practice sessions and observation during the trials. However, DARPA's many modifications of the task descriptions and our team's change from an initial goal of autonomous task execution to a teleoperation approach made the sensor head design less of a good fit in the end. For example, all of the head's capabilities for long-distance depth sensing for object detection and shape fitting became unnecessary with a human "oracle" to point out things in monocular images. Conversely, there was a lot of early concern about 3-D sensing for "very-near" grasping, such as ladder rungs and the roof pillars during vehicle ingress/egress, but the ladder subteam chose not to use this information and DARPA removed the ingress portion of the driving event.

Human teleoperation meant that we were trying to push relevant sensor data over a poor communications link instead of just to a local computer, making the effective framerates and resolutions of the sensors much lower and throwing away information. Cheaper and smaller sensors, and fewer of them, could probably have done the job just as well. The wide-angle lenses chosen for the cameras were excellent for scene context while driving or walking, but not so good for detail during fine grasping tasks. We considered having both wide- and narrow-angle stereo pairs like the PR2 [6, 11], but ultimately decided within the sensor head size and weight budget to use three cameras with identical lenses for multiple baselines instead. It is possible that by using motorized varifocal lenses for zooming we could have satisfied both requirements.

### 5.1.5 ROS

Our use of ROS on this project fell into two areas: operator tools and the perception computer. While it mostly enabled us to make rapid progress (see Section 5.2.4), at times our approach seemed fundamentally at odds with its design. For example, we were forced to literally sever the TF tree (the data structure maintaining relationships between various robot coordinate frames [17]) at the neck of the robot and subsequently write special-case code to stitch it back together, in order to prevent megabits of TF data from overwhelming our low-bandwidth communications link to the robot.

For us, one important lesson is that even well-written frameworks are not trivial to integrate. In future projects, we will make sure to explicitly plan adequate time to integrate outside code, and perform careful "impedance matching" to ensure that the assumptions on both sides of the API are met correctly.

### 5.1.6 Communications

Instead of obtaining the network device which was used at the DRC trials to shape communications traffic, we instead simulated the DRC network conditions with our own custom-written *TC* script [1] which imposed similarly if not more aggressive latency and bandwidth restrictions. Upon arrival to Homestead, however, it became clear that the extreme packet buffering and out-of-order delivery imposed by the DRC network caused our system some unforeseen difficulties, as neither phenomenon was modeled by our testing setup.

In some sense, we were aiming for the wrong sort of robustness. Our goal was to produce software that was functional across a range of latency and bandwidth conditions, but the reality had no range at all: just regular and predictable swapping between two network conditions according to a preset scheme. DARPA did participants a favor by describing in detail how the network at the DRC would be configured, and it was a mistake not to work from their specifications. One lesson for future challenges is to be careful about generalizing robustness when it is precisely specified in a task description.

### 5.1.7 Accelerating rate of hardware issues

As we neared the trials, we were puzzled by what appeared to be a dramatic drop in hardware reliability. In fact, we experienced more hardware failures in the final two weeks than we had during the entire month preceding them, ranging from burnt out motors, to motor control boards, to the main power distribution board itself. Looking back, it is clear that what changed was not the reliability of the hardware, but the amount of use. Simply put, as the software became more mature, we were just running the robot more. It is natural for robot duty cycles to peak towards the end of a project, and it is therefore vital to plan for peak use, not average.

Another lesson is to design good automatic safety systems. For example, DRC-HUBO's shoulder motors and motor control boards were susceptible to failure due to high currents and/or temperatures. Although we learned over time how to operate the robot in regimes that avoided burning out components, the more we operated the robot, the more chances we had to violate our own self-imposed guidelines. Counter-intuitively, system testing is likely to be less "safe" for hardware than nominal operation, and the system design should reflect that.

## 5.2 What went right

### 5.2.1 General-purpose, usable operator software

In the end, we were satisfied with the operator software we produced, both in that it faithfully and effectively exposed the low-level robot functionality that we developed, and in that it allowed a trio of operators to coordinate a 32-DOF humanoid robot in a number of challenging tasks with a single, unified approach.

We believe that the division of labor among the operators was an effective one in that it allowed each operator to be reasonably active at the same time without overwhelming any of them with cognitive load. During trajectory execution and monitoring, the perception manager can point the camera at the robot and/or obtain point clouds for the next trajectory. Furthermore, the trajectory designer can frequently begin designing the next trajectory during execution of the current one.

The system we produced was sufficiently general to handle three disparate events of the DRC trials: door opening, wall breaking, and debris removal. In fact, we are confident that our general-purpose approach would have been applicable to at least the valve turning and hose tasks as well. Focusing on a unified approach rather than developing specialized software for each event allowed us to perform more testing. Since all of the code was shared for the three events, a bug fix or enhancement which benefited our performance on one could potentially aid both other events as well.

### 5.2.2 Simplicity of implementations

We made two major design decisions in favor of simplicity which proved to be effective. The first was to eschew the use of complete planners for our subteam's tasks, and the second was to limit the use of complex feedback controllers onboard the robot.

Although both other DRC teams and other subteams on our own team used sampling-based planners such as CBi-RRT [5] to accomplish the DRC tasks, our design philosophy reflects the belief that the crux of the DRC is systems engineering in general, rather than motion generation in particular. Hence, despite the utility of complete planners, we preferred not to increase autonomy at the cost of additional system complexity. Such complexity could arise from explicitly modeling goal conditions unique to each event, as well as reasoning about shapes and affordances of objects in the environment (e.g. doors, debris, drill, etc.). Several of the manipulation strategies we elected to use would have been difficult to plan autonomously, especially those that exploited frictional properties of sliding contacts, which are notoriously difficult to model. Examples include sliding an arm across a door while pushing it open, or letting a piece of wood slide along the robot's hand to a safe drop point during debris removal.

Limiting ourselves to just a few vital onboard control schemes also proved helpful. Other than joint-level PD control, the only controllers we ran online were a simple impedance controller for balancing, and a gravity and friction compensation controller for compliant control of the arms (both described in Section 3.2). Each novel controller developed for a humanoid introduces concerns about convergence and stability, kinematic singularities, joint limits, and self-collisions, to name just a few. When multiple controllers may be active at the same time, there is also a combinatorial explosion of possible ways for them to interfere with each other. By making a principled decision about what controllers were truly necessary, we were able to reduce the overall complexity of the system.

A big advantage of the way we generated motion was that all trajectories were pre-generated with statically stable starting and ending poses at zero velocity. Each trajectory is generated and validated holistically – if the trajectory does not validate, the robot is safe because it remains in the statically stable starting pose. Again, we do not claim that this is the optimal approach, only that making these design decisions contributed valuable simplicity to our software implementation for this project.

### 5.2.3 Agile development

Starting around late October, we adopted several tenets of the agile development methodology [3, 25]. A whiteboard in the lab became dedicated to displaying a prioritized list of tasks, each on its own post-it note. Individuals on our subteam were allocated to the highest-priority task matching their skill set, and the list was reviewed weekly. During this phase of development, our goal was to maintain the end-to-end functionality of the system as features were added and bugs fixed.

Given the breadth of the DRC along with the fast schedule, we believe it would have been difficult to adhere to the agile method from the start. The tenet of maintaining an intact end-to-end system at all times is especially difficult at the start of large robotics projects, while developers experiment with different approaches, and the underlying capabilities of the hardware platform are only hazily understood. Still, we probably would have seen some benefit from switching to an agile approach slightly earlier than we did.

### 5.2.4 ROS

Overall, we found ROS to be a boon both for prototyping and for rapid development. Having pre-existing functionality for recording and transmitting image and point cloud data was a significant benefit and time saver, especially as we explored various implementation alternatives such as colored (XYZRGB) point clouds and multi-camera stereo. For the operator software and GUI, RViz also helped us make rapid progress by providing a plugin infrastructure upon which we could implement our own panes and visualizations.

### 5.2.5 Ach

Ach provided a very stable and low-latency IPC system onboard the robot, and the network forwarding Achd made it quite transparent to communicate to robot software from an operator computer. Compared with ROS, Achd adds less overhead in terms of both bandwidth and latency. During the final months of the project, we were able to ratchet down our communications bandwidth to the robot significantly due to two easily implemented changes: first of all, the author of Ach and Achd generously provided a throttling feature which capped the frequency of communications per-channel, so we could restrict robot state messages to arrive at no more than e.g. 2 Hz. Secondly, it is trivial to configure Achd to run over a compressed ssh connection via port forwarding. Combined, the two allowed us to reduce traffic between the robot computer and the operator computer from 2 Mbps to about 18 kbps with no observable reduction in functionality.

# 6 Conclusions and Future Work

Our high-level goal on this project was to produce a general-purpose system for teleoperation of the DRC-HUBO humanoid robot to address three events of the DRC trials, all of which demanded complex mobile manipulation capabilities. Although the particular implementation we produced exhibited some technical limitations, none of the problems we encountered were essentially attributable to the basic approach. Indeed, we believe it was a sound one overall in that it spared us from writing, for instance, a distinct special-purpose planner for each event, with task-specific domain knowledge encoded in each separate implementation.

If we were to undertake this project again from the beginning, there are certainly aspects we would choose to revisit differently. Combining competency-based, in addition to just event-based, task allocation during development would have improved our chances of success. Addressing common functionality from the beginning of the project and adhering to strong specifications whenever possible are also beneficial.

One key question, then, is whether we *would* in fact participate in another DRC-like project, given the opportunity. While some in the academic research community actively avoid these broadly scoped, competition-based programs, in our own experience we find that these integrated challenges stand as a powerful object lessons for researchers who typically work on isolated problems. Perhaps the biggest lesson for us was that the final system only looks as good as its weakest part. The most sophisticated manipulation planner in the world does no good if the robot cannot reliably and robustly walk to its destination. Nevertheless, we were glad to have the opportunity to compete at such a high level, and to work on a project which embodies the truly multidisciplinary nature of robots.

Aside from throwing into sharp relief the importance of sound systems engineering practices, the DRC has also played a large pedagogical role for the many students who worked on the project. Not only did our student team members accumulate more robot operation hours in a few months than many do over an entire robotics PhD program, but they also got to work with a large, integrated code base (our main code repository had over 80,000 physical lines of C++ code, as reported by SLOCCount [34]). Student team members reported that participation in this project has helped them develop skills in C++ coding, system design, software development, forward and inverse kinematics, real-time controls, and general principles of robot operations, to name just a few highlights. We believe that these large integrated challenges contribute to building a culture of competent generalists among our students.

### 6.1 Future work

Although we will not likely continue our efforts on the next phase of the DRC, we have plans to use the infrastructure we have built in future projects. We are currently working with KAIST and Rainbow, Inc. to obtain several upgrades to the DRC-HUBO itself, including stronger arm motors, better cooling, and increased sensing capacity. Aside from improving the reliability and robustness of legged locomotion, topics of ongoing research include discovering faster and less cognitively demanding ways of operating the robot. For the DRC, using a team of three expert operators was not overly burdensome; however, we are interested in improving the operator interface to the point where a single operator can perform useful tasks with minimal training. To that end, we are pursuing both higher-level behavior primitives providing more autonomy (e.g. "walk to object", "pick up object"), as well as more contextual awareness in the operator software.

## A Inverse Kinematics of the DRC-HUBO Arm

Input to the IK solver for the arms is $\mathbf{T}_{EE}^{P}$, the transformation from the end effector to the (root) pelvis frame of the robot, as well as the waist rotation angle $\varphi$ and the wrist roll angle $\omega$. Given these variables, we can compute $\mathbf{T}_{S}^{WP}$, the transformation from the

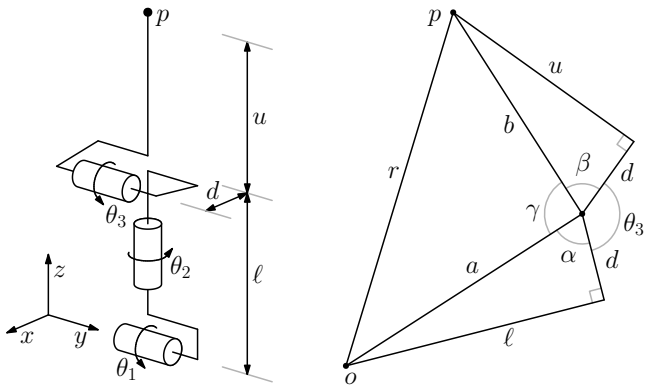


Figure 20: Left: 3D kinematics of wrist pitch ($\theta_1$), wrist yaw ($\theta_2$), and elbow pitch ($\theta_3$) joints. The shoulder is located at point $p$ in the frame of wrist pitch. Right: Planar view of the plane formed by the wrist, elbow, and shoulder. The law of cosines may be used to solve for the elbow angle $\theta_3$ using just the distance $r$ between the wrist and the shoulder.

shoulder to the wrist pitch joint:

$$\mathbf{T}_S^P = \left[\begin{array}{c|c} \mathbf{R}_z(\varphi) & \mathbf{t}_S \\ \hline 0 & 1 \end{array}\right] \tag{3}$$

$$\mathbf{T}_{EE}^{WP} = \left[\begin{array}{c|c} \mathbf{R}_x(\omega) & \mathbf{t}_{WP} \\ \hline 0 & 1 \end{array}\right] \tag{4}$$

$$\mathbf{T}_S^{WP} = \left(\mathbf{T}_{WP}^S\right)^{-1} = \mathbf{T}_{EE}^{WP}\ \left(\mathbf{T}_{EE}^P\right)^{-1}\ \mathbf{T}_S^P \tag{5}$$

Denote by $p$ the translation element of $\mathbf{T}_S^{WP}$. As illustrated on the left of Figure 20, it depends on the wrist pitch angle $\theta_1$, the wrist yaw angle $\theta_2$ and the elbow pitch angle $\theta_3$, as well as the link lengths $\ell, u$, and $d$:

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{R}_y(\theta_1)\,\mathbf{R}_z(\theta_2)\left(\begin{bmatrix} d \\ 0 \\ \ell \end{bmatrix} + \mathbf{R}_y(\theta_3)\begin{bmatrix} -d \\ 0 \\ u \end{bmatrix}\right) \tag{6}$$

Expressing $x, y$ and $z$ in terms link lengths and sines and cosines of the three angles, we have

$$x = ds_1s_3 + dc_1c_2 - dc_1c_2c_3 + \ell s_1 + us_1c_3 + uc_1c_2s_3 \tag{7}$$

$$y = ds_2 - ds_2c_3 + us_2s_3 \tag{8}$$

$$z = dc_1s_3 - ds_1c_2 + ds_1c_2c_3 + \ell c_1 + uc_1c_3 - us_1c_2s_3 \tag{9}$$

We solve the angles in reverse order, starting from the elbow angle $\theta_3$. Inspecting the plane containing the wrist, the elbow and the shoulder (as depicted on the right side of Figure 20), we observe that the length $r$ of the vector $p$ depends only upon the elbow angle. The law of cosines gives the solution for $\theta_3$:

$$
\begin{aligned}
a^2 &= \ell^2 + d^2 && (10)\\
b^2 &= u^2 + d^2 && (11)\\
r^2 &= x^2 + y^2 + z^2 && (12)\\
\alpha &= \arctan(\ell/d) && (13)\\
\beta &= \arctan(u/d) && (14)\\
\gamma &= \arccos\left(\frac{r^2 - a^2 - b^2}{2ab}\right) && (15)\\
\theta_3 &= 2\pi - \alpha - \beta - \gamma && (16)
\end{aligned}
$$

Once $\theta_3$ is known, (8) may be solved straightforwardly to obtain $\theta_2$:

$$
\theta_2 = \arcsin\left(\frac{y}{d - dc_3 + us_3}\right) \quad (17)
$$

Finally, we observe that (7) and (9) above are linear in $c_1$ and $s_1$. Rather than solving for either one individually, we use the arctangent of their ratio, which is more numerically stable, and gives an answer in the correct quadrant when the $atan2$ library function is used to compute $\theta_1$.

$$
\begin{aligned}
f &= dc_2 - dc_2c_3 + uc_2s_3 && (18)\\
g &= \ell + ds_3 + uc_3 && (19)\\
x &= fc_1 + gs_1 && (20)\\
y &= gc_1 - fs_1 && (21)\\
\theta_1 &= \arctan\left(\frac{fx + gy}{gx - fy}\right) && (22)
\end{aligned}
$$

Once $\theta_1$, $\theta_2$, and $\theta_3$ are computed, the joint angles for the spherical shoulder joint may be computed using a rotation matrix to Euler angle conversion with the appropriate sequence. Up to four distinct solutions for the lower arm may be computed by flipping the sign of $\gamma$ in the equations above, and/or by choosing an alternate $\theta_2$, offset by $\pi$. Combined with a shoulder flip, this allows up to eight distinct solutions (however, typically few of them are within joint limits or near the previous joint configuration).

## Acknowledgments

This project was funded by DARPA N65236-12-1-1005: DARPA Robotics Challenge, with additional funding from NSF CNS-0960061 MRI-R2: Unifying Humanoids Research.

The authors wish to thank Dr. Magnus Egerstedt of GA Tech, as well all of our student contributors: Peter Vieira, Eric Huang, Andrew Price, Juan Carlos Garcia, Neil Dantam, Ana Huamán Quispe, Rowland O'Flaherty, Krunal Chande, Keliang He, Beth Martin, Gene "Temple" Price, and Will Schneider.

## References

[1] tc(8): show/change traffic control settings - Linux man page. http://linux.die.net/man/8/tc. Accessed: February 2014.

[2] The DARPA Robotics Challenge. http://www.theroboticschallenge.org/. Accessed: February 2014.

[3] Agile Alliance. Manifesto for agile software development. http://agilemanifesto.org/, 2001. Accessed: 2013-12-10.

[4] N. Alunni, C. Phillips-Grafftin, H. B. Suay, D. Lofaro, D. Berenson, S. Chernova, R. W. Lindeman, and P. Oh. Toward a user-guided manipulation framework for high-DOF robots with limited communication. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, pages 1–6. IEEE, 2013.

[5] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner. Manipulation planning on constraint manifolds. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 625–632. IEEE, 2009.

[6] J. Bohren, R. B. Rusu, E. G. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mosenlechner, W. Meeussen, and S. Holzer. Towards autonomous robotic butlers: Lessons learned with the PR2. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 5568–5575. IEEE, 2011.

[7] M. Buehler, K. Iagnemma, and S. Singh. *The 2005 DARPA Grand Challenge*. Springer, 2007.

[8] M. Buehler, K. Iagnemma, and S. Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*. Springer, 2009.

[9] J. Chestnutt. *Navigation Planning for Legged Robots*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2007.

[10] J. Chestnutt, M. Lau, G. Cheng, J. Kuffner, J. Hodgins, and T. Kanade. Footstep planning for the Honda ASIMO humanoid. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, Barcelona, Spain, April 2005.

[11] S. Cousins. ROS on the PR2. *Robotics & Automation Magazine, IEEE*, 17(3):23–25, 2010.

[12] H. Dang, Y. Jun, P. Oh, and P. K. Allen. Planning complex physical tasks for disaster response with a humanoid robot. In *Proc. IEEE Int'l Conf. on Technologies for Practical Robot Applications*, pages 1–6. IEEE, 2013.

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

[14] DARPA. DRC Trials Initial Task Descriptions DISTAR Case 21473. http://www.theroboticschallenge.org/.

[15] DARPA. DRC Trials Task Description Release 11 DISTAR 22197. http://www.theroboticschallenge.org/. Accessed: 2013-12-10.

[16] D. Fiser. libccd - Library for collision detection. http://libccd.danfis.cz/. Accessed: February 2014.

[17] T. Foote. tf: The transform library. In *Proc. IEEE Int'l Conf. on Technologies for Practical Robot Applications*, pages 1–6. IEEE, 2013.

[18] M. Grey, N. Dantam, D. M. Lofaro, A. Bobick, M. Egerstedt, P. Oh, and M. Stilman. Multi-process control software for HUBO2 plus robot. *Proc. IEEE Int'l Conf. on Technologies for Practical Robot Applications*, 2013.

[19] D. Hackett, J. Pippine, A. Watson, C. Sullivan, and G. Pratt. The DARPA Autonomous Robotic Manipulation (ARM) program: a synopsis. *Autonomous Robots*, 36(1-2):5–9, 2014.

[20] L. D. Jackel, E. Krotkov, M. Perschbacher, J. Pippine, and C. Sullivan. The DARPA LAGR program: Goals, challenges, methodology, and phase I results. *Journal of Field Robotics*, 23(11-12):945–973, 2006.

[21] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE International Conference on Robotics and Automation*, 2003.

[22] J. Kim, I. Park, J. Lee, M. Kim, B. Cho, and J. Oh. System design and dynamic walking of humanoid robot KHR-2. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2005.

[23] J. Kim, S. W. Park, I. W. Park, and J. H. Oh. Development of a humanoid biped walking robot platform KHR-1: Initial design and its performance evaluation. In *Proc. Int'l Workshop on Humanoid and Human Friendly Robotics*, pages 14–21, 2002.

[24] M. Kleine-Budde. SocketCAN–The official CAN API of the Linux kernel. *Proceedings of the 13th iCC*, 2012.

[25] C. Larman. *Agile and iterative development: a manager's guide*. Addison-Wesley Professional, 2004.

[26] J. Oh, D. Hanson, W. Kim, I. Han, J. Kim, and I. Park. Design of android type humanoid robot Albert HUBO. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2006.

[27] J. Oh, D. Hanson, W. Kim, I. Han, J. Kim, and I. Park. Practical experiment of balancing for a hopping humanoid biped against various disturbances. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 2010.

[28] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

[29] I. Park, J. Kim, J. H. Lee, , and J. Oh. Mechanical design of humanoid robot platform KHR-3 (KAIST humanoid robot 3: HUBO). In *Proc. IEEE-RAS Int'l Conf. on Humanoid Robots*, 2005.

[30] I. Park, Y. Kim, S. Park, and J. Oh. Development of humanoid robot platform KHR-2 (KAIST humanoid robot 2). In *Proc. IEEE-RAS Int'l Conf. on Humanoid Robots*, 2004.

[31] C. Phillips-Griffin. WPI-ARC/teleop_toolkit. https://github.com/WPI-ARC/teleop_toolkit. Accessed: February 2014.

[32] J. Pippine, D. Hackett, and A. Watson. An overview of the Defense Advanced Research Projects Agency's Learning Locomotion program. *The International Journal of Robotics Research*, 30(2):141–144, 2011.

[33] 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. ICRA workshop on Open-Source Software*, 2009.

[34] D. A. Wheeler. SLOCCount. http://www.dwheeler.com/sloccount/sloccount.html. Accessed: February 2014.

[35] Y. Zhang, J. Luo, K. Hauser, R. Ellenberg, P. Oh, H. A. Park, and M. Paldhe. Motion planning of ladder climbing for humanoid robots. In *Proc. IEEE Int'l Conf. on Technologies for Practical Robot Applications*, pages 1–6. IEEE, 2013.

[36] Y. Zheng, H. Wang, S. Li, Y. Liu, , D. Orin, K. Sohn, Y. Jun, and P. Oh. Humanoid robots walking on grass, sands, and rocks. In *Proc. IEEE Int'l Conf. on Technologies for Practical Robot Applications*, 2013.