A General-purpose System for Teleoperation of the DRC-HUBO Humanoid Robot

Matt Zucker
Department of Engineering, Swarthmore College, Swarthmore, Pennsylvania 19081
e-mail: mzucker1@swarthmore.edu
Sungmoon Joo and Michael X. Grey
School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia 30332
e-mail: sungmoon.joo@gatech.edu, mxgrey@gatech.edu
Christopher Rasmussen
Department of CIS, University of Delaware, Newark, Delaware 19716
e-mail: cer@cis.udel.edu
Eric Huang, Michael Stilman, and Aaron Bobick
School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia 30332
e-mail: ehuang3@gatech.edu, mstilman@cc.gatech.edu, afb@cc.gatech.edu

Received 1 March 2014; accepted 23 November 2014

We present a general system with a focus on addressing three events of the 2013 DARPA Robotics Challenge (DRC) 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. © 2015 Wiley Periodicals, Inc.

1. INTRODUCTION

Previous DARPA-sponsored competition programs in robotics have promoted research in topics as diverse as learning for ground vehicles (Jackel et al., 2006), legged locomotion (Pippine, Hackett, & Watson, 2011), and autonomous manipulation (Hackett et al., 2014); however, the 2013 DARPA Robotics Challenge (DRC) is most similar in scope to the 2004–2005 Grand Challenge and the 2007 Urban Challenge, which helped bring about a revolution in the field of autonomous driving (Buehler, Iagnemma, & Singh, 2007, 2009). Like these broadly scoped challenges, the DRC is aimed at producing robotic systems that integrate expertise from nearly every subdiscipline of robotics, and it presents valuable opportunities to apply both established and novel research methods to real-world scenarios.

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. The 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, and by allowing operators to coordinate high-level behavior through interpolation between key poses of the robot, which respects balance and pose constraints.

The authors of this paper form just a small subteam of the multi-institution DRC-HUBO team, led by Dr. Paul Oh of Drexel University. Due to the team's distributed nature, participating institutions each formed subteams focused on one or more individual DRC events. For information about our colleagues' approaches, we refer the reader to their existing publications related to the DRC (Alunni et al., 2013; Dang, Jun, Oh, & Allen, 2013; Zhang et al., 2013; Zheng et al., 2013).

The remainder of this paper is organized as follows: in Sections 2 and 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.

Direct correspondence to: Sungmoon Joo, sungmoon.joo@gatech.edu

Table I. 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

2. HARDWARE

DRC-HUBO's hardware design is based on its immediate predecessor, HUBO2+, developed at KAIST. The first generation, KHR-1, debuted in 2002, and there have been several platform upgrades since then (Kim, Park, Park, & Oh, 2002; Zheng et al., 2013). The development of the HUBO series is summarized in Table I. DRC-HUBO is the result of a collaboration between the Drexel-led DRC-HUBO team, KAIST, and Rainbow, Inc., based on our team's analysis of the performance of the HUBO2+ on early versions of the DRC task descriptions (DARPA, 2013). Once 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, highlighting the major changes from HUBO2+.

2.1. DRC-HUBO Hardware Overview

DRC-HUBO is 1.47 m tall with a wingspan of 2.04 m, it weighs 52 kg (including battery), and it has 32 degrees of freedom (DOF). Relative to its predecessor, it was given longer (15% increase for legs and 58% for arms) and stronger limbs with an aluminum skeleton and shell, resulting in a taller and heavier robot. As Figure 1 shows, 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 (DARPA, 2013).

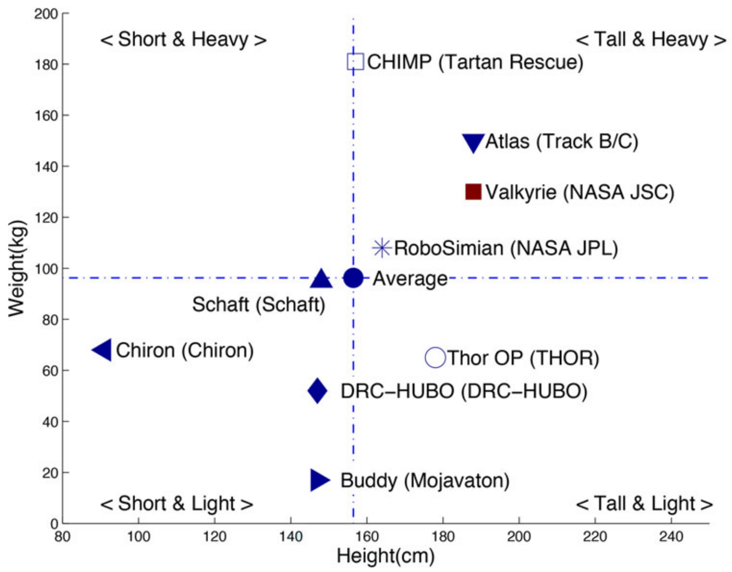
The DRC-HUBO upgrade increased the arm from six to seven DOF, and grasping capabilities were upgraded as well. Three fingers on each hand close via a single motor for power grasps. On the right hand, there is an additional trigger finger that moves independently, allowing the robot to operate power tools. The fingers can support up to approximately 9 kg. DRC-HUBO's joint motor controllers were also upgraded to add two new control modes: complementary switching and noncomplementary switching. In noncomplementary 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), allowing safe interaction between DRC-HUBO and the environment.

2.2. Perception System

We designed and built the robot's sensor head. It pans $\pm 180^\circ$ and tilts $\pm 60^\circ$ without self-collision, and it has the following sensors:

- **3 × Point Grey Flea3 cameras**, each with $1,280 \times 1,024$ resolution and approximately $90^\circ \times 70^\circ$ field of view (FOV), forming a synchronized stereo rig with baselines of 6, 12, and 18 cm.
- **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 detectable depth ranges from 0.1 to 30 m, and reflectance information is provided for each point. The lidar is mounted on a dedicated tilting servo that has a range of $\pm 60^\circ$.
- **Microstrain 3DM-GX3-45 IMU** with three-axis accelerometer, three-axis gyro, and GPS.
- **PrimeSense short-range RGB-D camera** (rear-facing), which captures RGB and depth images at 640×480 resolution with a FOV of $57.5^\circ \times 45^\circ$. The PrimeSense has a depth range from 0.35 to 1.4 m, but it does not work well in direct sunlight.

Perceptual tasks common to all of the events were analyzed 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 the following categories: *long-distance* (> 5 m away), including landmark, object, and obstacle detection for purposes of setting walking and driving navigation goals; *mid-distance* (1–5 m away), including detailed terrain characterization for imminent obstacle avoidance and footstep planning; and *near-distance* (< 1 m), including object characterization and pose measurement for planning and monitoring grasping motions. Sensors were selected to collectively provide high-resolution three-dimensional (3D) 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 are available at high frame-rate, coarse depth several times per second, and fine depth once every several seconds, with sufficient redundancy to complete tasks in the presence of sensor failures.



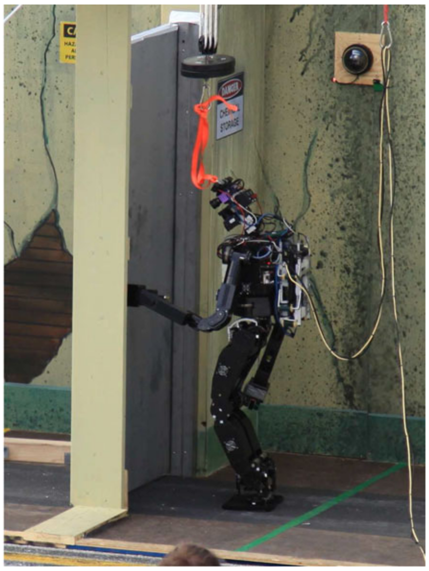


Figure 1. Left: Comparison of robot platforms in the DRC trials. Right: DRC-HUBO participating in door opening event at DRC trials.

3. SOFTWARE

The overarching software design is based on a distributed multiprocess architecture that provides robustness and modularity. Robustness is achieved by designing processes to gracefully handle interruptions in communication or operation. In the event of an unexpected failure, components are designed to behave reasonably and bring the system to a safe state while recovery is attempted. Modularity is achieved through a publish/subscribe framework in which individual processes can be readily modified or replaced without adversely affecting the overall pipeline. The multiprocess design also effectively distributes the computational load between the robot's onboard computers and the operator workstations. More detail on the design philosophy and motivation, especially during the early stages of development, is available in Grey et al. (2013).

The diagram in Figure 2 provides an overview of the connections between key processes in the system. Processes on the operator workstations run on top of the robot operating system (ROS) framework (Quigley et al., 2009), whereas onboard processes[1] communicate via Ach, an efficient IPC protocol designed for real-time robotic applications (Dantam & Stilman, 2012). Processes onboard the robot are robust to intermittent communication, and run asynchronously from the processes on the operator workstations. Their purpose is to perform real-time control on top of the whole-body trajectories provided by the operators. Processes on the operator workstations aid the operators in constructing trajectories, provide the operators with situational awareness, and forward commands to the robot. Unlike the onboard processes, they are not subject to real-time constraints or CPU usage limitations.

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

3.1. Low-level Software

The low-level DRC-HUBO software communicates with the hardware through a controller area network (CAN) bus. The control frequency of the robot is 200 Hz, constrained chiefly by CAN bandwidth limitations. The onboard operating system is an Ubuntu 12.04 Server with the Preempt-RT patch applied, and processes are 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 executing them via real-time closed-loop control (discussed in Section 3.2). All trajectories begin and end in statically stable states, and the entire trajectory must be received before execution begins. Upon completion of a trajectory, the trajectory executor continues to maintain balance and compliance control on the final robot configuration until the next trajectory is provided.

3.2. Real-time Feedback Controllers

Our system runs two feedback controllers online: balance control and compliance control. The leg and hip joints are used to balance the robot by shifting its pelvis, while the arms achieve compliance through DRC-HUBO's

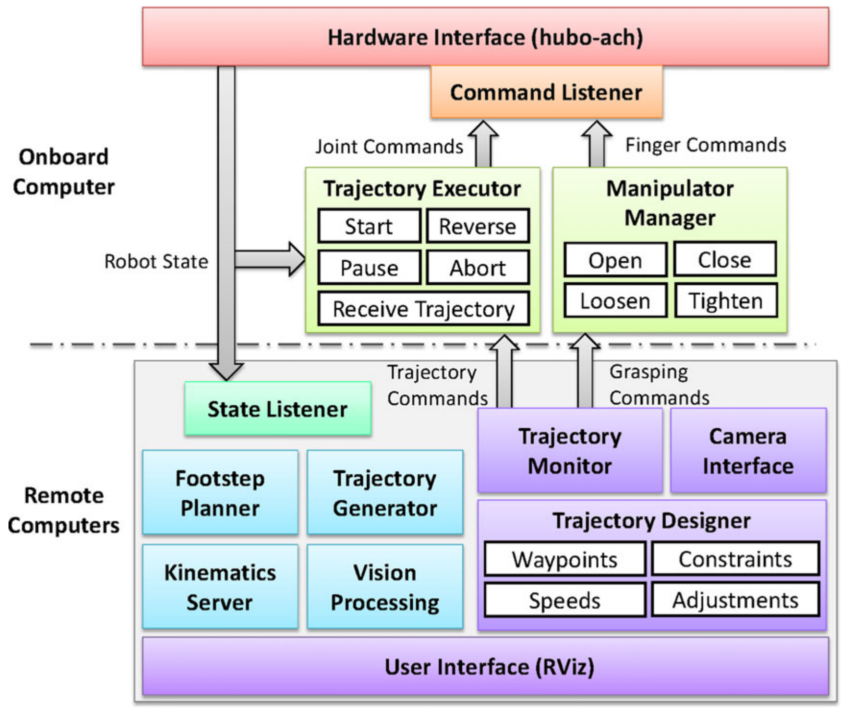


Figure 2. Diagram of our software architecture. Boxes indicate independent processes, with colors ranging the spectrum from red (hardware interface) to purple (human interface). Gray arrows are Ach channels, and the large gray box represents the ROS framework.

noncomplementary switching mode. The goal of the balance controller is to correct for model error or unpredicted external forces during quasistatic motion and manipulation, whereas the objectives of the compliance controller are 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, which is used to measure the robot's zero moment point (ZMP). Comparing it to a desired ZMP provides an error vector that indicates the direction for the robot's pelvis to maintain balance. Our balance 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 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. Given $\mathbf{s}$, we use IK (see Section 3.3) to convert the Cartesian offset for the pelvis to a joint offset for each leg joint.

3.2.2. Compliance Control

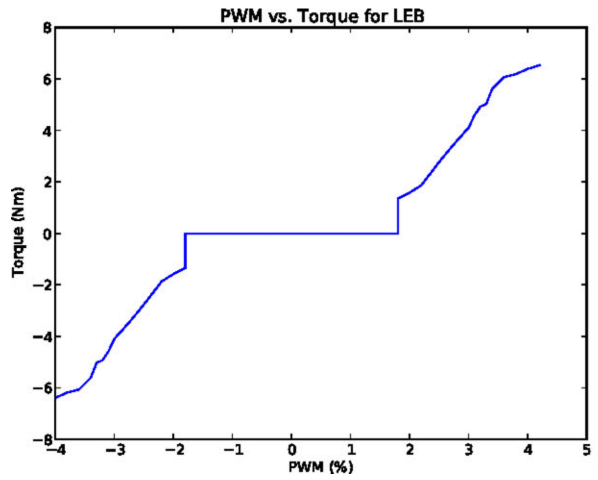
DRC-HUBO's noncomplementary switching mode allows our software to control the joints using pulse width modulation (PWM) commands, as opposed to raw position commands. Although PWM does not directly map to torque, we were able to construct a rough empirical relationship between the two (see Figure 3) suitable for implementing the torque control law:

$$\tau_i = K_{pi}\,(\theta_{di} - \theta_i) - K_{di}\,\dot{\theta}_i + \tau_{Gi}(\theta).$$

Here, τ_i is the commanded torque for joint i, θ is a vector of joint angles, θ_{di} is the desired angle of joint i, θ_i is the measured angle, $\dot{\theta}_i$ is the measured velocity, and $\tau_{Gi}(\theta)$ is the computed torque due to gravity at configuration θ. This control law corresponds to low-gain PD control with feedforward gravity compensation.

Many joints have a deadband in which a nonzero PWM command results in approximately zero torque output. This can be mainly attributed to friction in the joint driving system, and it requires special compensation to overcome. Our overall mapping from commanded torque to PWM is given by

$$V_i = T_i^{-1}(\tau_i) + \min\left(K_{fi}\,\dot{\theta}_i, \operatorname{sgn}(\dot{\theta}_i)V_{f\max,i}\right),$$



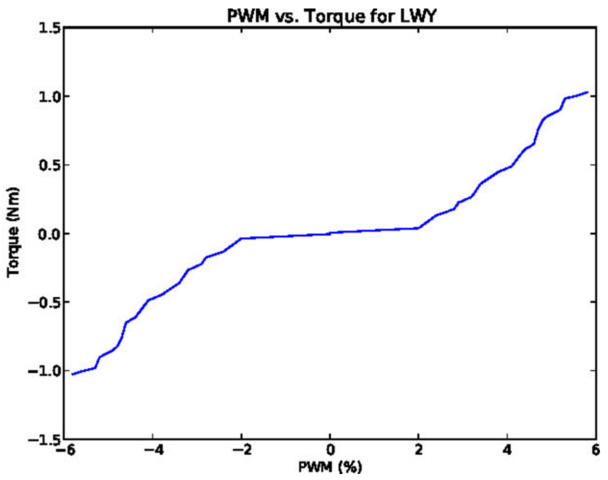


Figure 3. Empirical PWM vs torque relationship for left elbow (LEB) and left wrist yaw (LWY).

where V_i is the PWM output, and $T_i^{-1}(\tau_i)$ is the inverse of the PWM-torque relationship plotted in Figure 3. The right-hand term provides a small boost in the direction of the joint's measured velocity, capped to the size of the joint's deadband $V_{f_{\max},i}$. Combined, these two control laws provide reasonably precise compliance control of the arms.

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

3.3.1. Kinematics and Limb IK

The DRC-HUBO robot has kinematically redundant, 7-DOF arms, comprised of a spherical shoulder, an elbow, and a spherical wrist. Each 6-DOF leg is comprised of a spherical hip, a knee, and ankle pitch and roll joints. The IK for both the arms and the legs is obtained analytically. The IK solver for the arms assumes a fixed wrist roll (the final joint in the series leading from shoulder to end effector), and it solves for the remaining six degrees of freedom. In our operator software, we effectively use the wrist roll 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 to avoid collisions.[2]

3.3.2. State Representation

Our trajectory generation system uses a redundant representation of a robot state, given by

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

where θ is the vector of joint angles and $\mathbf{T}_P$ 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 subsequent $\mathbf{T}_e$ is a rigid body transformation representing the position and orientation of an end effector e. The control modes $M_e \in \{joint, body, world\}$ indicate whether a limb is currently being controlled at the 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 θ to correspond to the transformation $\mathbf{T}_e$ of each end effector e, given the pose $\mathbf{T}_P$ of the root of the kinematic tree. During state resolution, 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 $\mathbf{T}_e$ in the body frame. Finally, if the limb is in $world$ mode, the desired effector pose in the body frame is given by $\mathbf{T}_P^{-1}\mathbf{T}_e$ and the joints for that limb are determined accordingly by the IK solver. The previous joint angles in the θ vector are used to disambiguate between multiple IK solutions, always favoring the solution closest to the previous configuration.

[2]Although our IK is analytical and not Jacobian-based, we use "null space" here in the sense of generating motions (such as orbiting the elbow) that leave the end-effector pose fixed.

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 splineinterpolation on angles or rigid body transformations, *resolve* performs whole-body IK, and *validate* checks forcontinuity and self-collisions.

```
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 $\mathbf{v}_e, \mathbf{a}_e$ for each end effector $e$, 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 $M_e$ 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 e do
        place $\mathbf{T}_{init,e}$ in the same frame as $\mathbf{T}_{goal,e}$ as indicated by $M_{goal,e}$;
        $t_{max} \leftarrow \max\left(t_{max},\ duration(\mathbf{T}_{init,e}, \mathbf{T}_{goal,e}, \mathbf{v}_e, \mathbf{a}_e)\right)$;
    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
        $u \leftarrow i/n$;
        for each joint j do $\theta_{i,j} \leftarrow smoothstep(\theta_{init,j}, \theta_{goal,j}, u)$;
        for each end effector e do $\mathbf{T}_{i,e} \leftarrow smoothstep(\mathbf{T}_{init,e}, \mathbf{T}_{goal,e}, u)$;
        $\mathbf{T}_{i,P} \leftarrow smoothstep(\mathbf{T}_{init,P}, \mathbf{T}_{goal,P}, u)$;
        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
```

3.3.3. Whole-body and Center of Mass IK

When resolving a state, whole-body 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. 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

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

where $\mathbf{x}(\mathbf{q})$ denotes the computed center of mass given the current state, and α is a step size (we find that $\alpha \approx 0.5$ gives very fast convergence). In addition to driving the COM to a point, we can just as easily drive it to a region such as a conservative approximation to the robot's 2D support polygon on the ground.

Although we implemented code to compute the whole-body COM Jacobian given the end-effector constraints, we found that the naive 2D update rule above was faster in practice. In benchmarks averaged across 300 regularly spaced robot positions, the 2D method outperformed whole-body steepest descent by a factor of 2 (1.7 vs 3.4 ms per query, on average); however, one drawback of our translation-only strategy vs a whole-body Jacobian method is that it fails to exploit all degrees of freedom of the robot (e.g., tilting the body to balance).

3.3.4. Interpolation and Trajectory Validation

Trajectories in our system are generated via smooth interpolation between two statically stable poses of the robot that are free of self-collisions. The algorithm for interpolation is detailed in Algorithm 1. All trajectories are represented both in memory and on the network as a sequence of joint configurations sampled at the control frequency of the robot ($\Delta t = 5$ ms), and augmented with additional metadata, including whether the balance and compliance controllers (Sections 3.2.1 and 3.2.2) should be active, as well as the desired location of the COM or ZMP.

As Algorithm 1 shows, during trajectory generation we first interpolate between robot states in joint space, followed by interpolating any rigid transformations, before finally resolving the state. Interpolating joint angles maintains continuity in angles that 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 all move smoothly in the workspace.

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, or the robot is found to be in self-collision. For collision checking, we represent the robot as a union of simple convex geometric objects such as capsules and boxes, as illustrated in Figure 4. For safety, the collision volumes are enlarged slightly beyond the actual physical dimensions of the robot. The *libccd* library (Fiser, 2010) is used for collision detection due to its improved speed compared to traditional triangle mesh representations.[3] In an experiment averaged across 20,000 randomly generated joint configurations, our *libccd*-based checker took 0.13 ms per query, whereas triangle mesh checking with the state-of-the-art FCL library (Pan, Chitta, & Manocha, 2012) took 0.45 ms. Of these 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. Overall, trajectory generation is responsive enough to provide fast feedback for operators. For example, on a sample run of the debris clearing task, 38 trajectories were generated. Average trajectory generation time was 5.8 s per trajectory with a standard deviation of 1.2 s.

Although our interpolated trajectories are not guaranteed to be dynamically stable, in practice the online balancing controller (Section 3.2.1) is effective over a large range of acceleration and velocity profiles. We observed few balancing problems stemming from the motion of trajectories themselves, as opposed to forceful interactions with the environment.

3.4. Walking

We implemented a dynamically stable walking trajectory generator using a ZMP preview controller to generate whole-body trajectories (Kajita et al., 2003). Operators can generate any of a number of regular walking gaits, including walking forward/backward, turning left/right, and sidestepping left/right. Additionally, we produced a footstep planner, similar to Chestnutt et al. (2005) and Chestnutt (2007), capable of generating a sequence of footsteps to bring the robot to an arbitrary goal position and orientation (see Figure 4).

Currently, the ZMP preview controller and the key pose interpolation described above are the only means of generating motion for our system. Although we have implemented others, such as direct teleoperation via haptic devices, we decided to limit ourselves to a smaller set of functionality to preserve simplicity.

3.5. Operator Tools and Communications

Based on prior research in search and rescue robotics showing the advantages of multioperator approaches (Burke & Murphy, 2004), we divided the operator tasks into three main roles. The *trajectory designer* is responsible for construction and sequencing of key poses that are connected via 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 as plugins for the the ROS RViz program.

Instead of developing functionality to explicitly the model and/or recognize objects at run-time, we instead use the human operators' ability to interpret camera and lidar point cloud data. Furthermore, our system requires neither global localization nor mapping, since our dense sensing allows us to build a relatively rich representation of the environment at all times. We developed a library of key poses for each task (see Figure 6), based on 3D virtual mock-ups of the events created from the task descriptions (DARPA, 2013). For example, in the debris removal task, we were able to preselect the general robot configurations for pregrasp, grasp, lift, and drop phases for each piece of debris.

The division of labor between the three roles allows operators to parallelize the work of supervising the robot. In three practice runs rehearsing the debris clearing task, for example, the trajectory designer was active for an average of 354 s, the execution manager for 430 s, and the perception manager for 53 s. The trials, illustrated in Figure 5, highlight a key trend we observed: operation times for the trajectory designer on a particular task tend to decrease with practice, whereas the execution manager's operation time remains fairly constant. Independent of the effect of practice, we note that a single operator would have to be active for a total of about 14 min in order to accomplish the same work as the trio, leaving far less time for actual trajectory execution on the robot.

3.5.1. Trajectory Designer

Our operator tools provide interactive markers that allow the trajectory designer to manipulate the robot's feet, hands,

[3]We also selected *libccd* due to its support of fast and accurate separation distance and penetration distance queries; however, we ended up focusing on collisions only in this work.

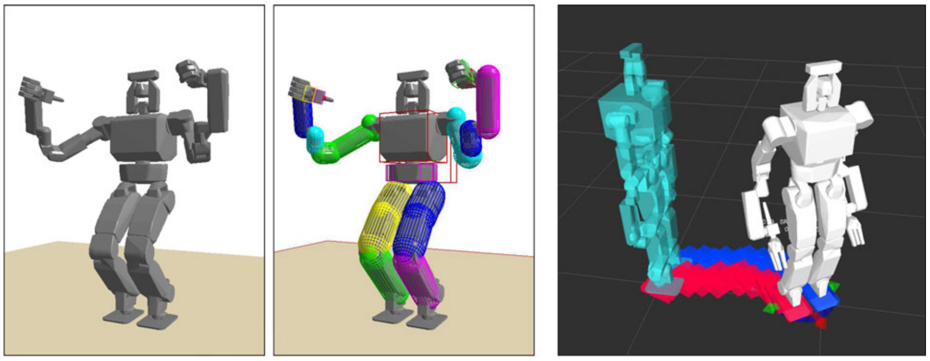


Figure 4. Left and middle: Triangle mesh model of DRC-HUBO; conservative collision geometry created from simple convex geometric objects, allowing efficient detection of self-collisions. Right: Our footstep planner generates walking trajectories to a desired destination. The translucent cyan model is the current state, the white model is the planned state. An interactive marker (arrows) allows the operator to specify a walking destination.

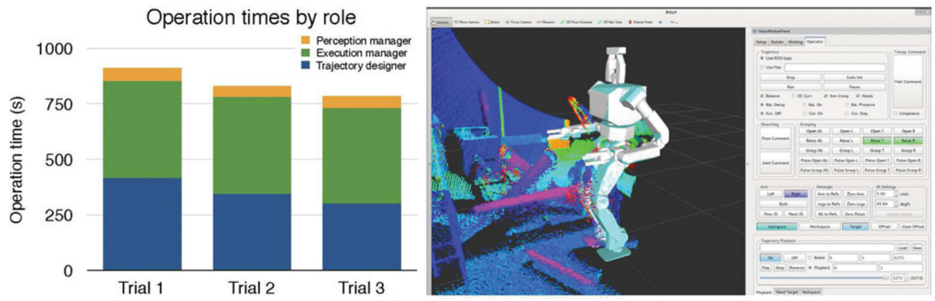


Figure 5. Left: Graph of operation times in three practice runs of the debris clearing task. Right: Screen shot of execution manager interface in RViz (see Figure 6 for the trajectory designer interface).

and pelvis. Separate controls on a dockable panel provide functionality to adjust the waist angle and wrist roll angles, as well as to modify end effector poses numerically. The operator tools use the COM IK procedure (see Section 3.3.3) to ensure static stability of every key pose, and they prevent the operator from generating any pose that places the robot into self-collision, or that violates joint limits.

In addition to the configuration of the robot, the trajectory designer selects for each destination key pose the desired interpolation mode (*joint*, *body*, or *world*) as well as the speed and acceleration limits for joints and end effectors. Although the default limits are reasonable for a wide variety of motions, the operator may choose to slow down during complex manipulation procedures or to assist the balance controller when manipulating heavy objects. During operation, the trajectory designer selects an appropriate key pose from the library and modifies it to reflect the environment and perception sensor readings (see Figure 7). For instance, the designer may modify the planned end-effector pose after walking to an object in order to more accurately grasp its target. The robot's current pose may be joined with one or more key poses into a trajectory via interpolation. The trajectory designer may specify walking destinations for the robot by dragging the interactive foot marker (see Section 3.4 and Figure 4). Walking trajectories are typically generated 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. All generated trajectories may subsequently be sent to the execution manager's workstation via ROS.

3.5.2. Execution Manager

The execution manager is responsible for sending trajectories to the robot and monitoring their execution. In the case of errors such as collisions with obstacles or failed grasps, the execution manager may pause trajectory execution and determine an appropriate course of action. Minor errors can be corrected by refining the key pose specifying the end point of the trajectory using interactive markers, whereas major ones will likely require attention from the trajectory designer. The execution manager is also responsible for taking small footsteps to correct the robot's approach if it does not arrive precisely at its destination due to accumulated errors in walking odometry.

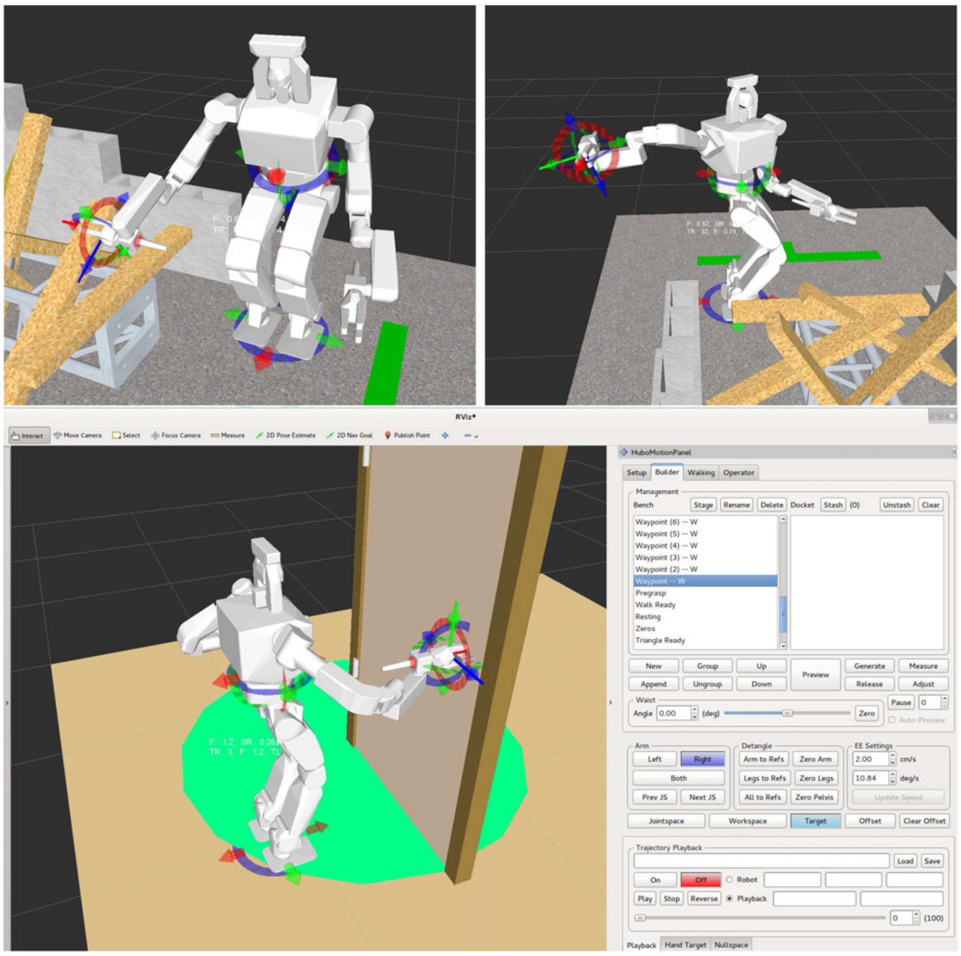


Figure 6. Virtual mock-ups of DRC events aided in building a library of key poses. Top: Sample key poses for the debris removal task. Left: Grasping a diagonally oriented piece of debris. Right: Preparing to safely drop a board. With this hand orientation the wood will slide down through the fingers safely behind the robot. Such poses are easy for the human operator to provide, but difficult to encode in a general manner for autonomous systems. Bottom: Key pose for door opening task, along with a subset of our RViz user interface. The interactive markers (arrows on the left-hand side) allow the operator to position the robot's feet, pelvis, and hands.

A further responsibility of the execution manager is to determine which controllers should be active during each trajectory execution. To prevent buildup of internal forces, the balance controller is disabled when pushing or pulling fixed objects in the environment. 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 lightweight, we instead ran the arms using the more precise stiff proportional derivative (PD) controller. Finally, the execution manager is responsible for operating the robot's hands.

Since there is no force or pressure sensing at the fingers, the execution manager uses visual feedback from the camera and the lidar to supervise grasping.

Although some responsibilities such as walking and trajectory refinement are shared by the trajectory designer and the execution manager, we 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 data necessary to enable the other two operators to perform their jobs well. The sensor head's orientation is controlled by the perception manager, independent of the rest

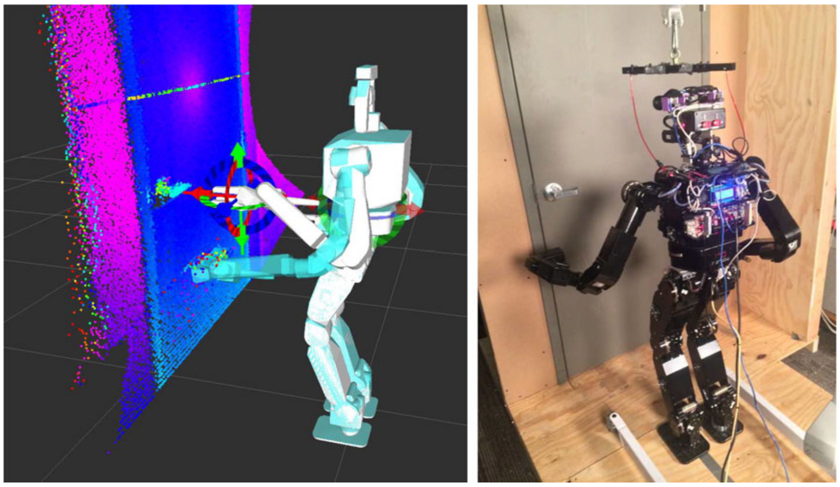


Figure 7. Left: Screen shot of an operator's view in RViz during the door opening task. The transparent cyan robot model is the actual robot state, the solid white model is the operator's puppet, and the point cloud is a lidar scan of the door. Right: the actual robot and door.

of the robot's joints. The perception manager also supervises communications bandwidth and 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 autoexposure can also be adjusted on the fly.

Unlike camera images that are streamed continuously, lidar point cloud data are collected from the robot only at the request of 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 interpreting 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 2 Hz to minimize bandwidth, and trajectories are sent to the robot via Achd as well. To conserve bandwidth, compressed camera images and point clouds (via JPG and zlib, respectively) are sent to the perception manager workstation from the robot via ROS, using WPI's *teleop_toolkit* for image transport (Phillips-Griffin, 2013).

Steady-state bandwidth for the entire system is under the 100 kbps lower limit imposed at the DRC trials events. Periodic robot state messages consume 18 kbps, and the perception computer on the robot consumes 47 kbps transmitting to the operator, with a 7 kbps stream of communications in the opposite direction. Transmissions of lidar point clouds and trajectories are not included in these bandwidth numbers. Each type of message is sufficient to dominate the low-bandwidth communications link when transmitted; however, neither one is streamed continuously.

4. LAB TESTS AND DRC TRIALS

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 that represented the DRC trial conditions and rules available at the time as accurately as possible.

Each of the three DRC trial events described below must be completed in a maximum of 30 min (minus a time penalty of 5 min per human intervention), with an additional 15 min of setup time provided beforehand. Generally, up to three points can be earned for subtasks in each event, with an additional bonus point for completing all subtasks without human intervention. See DARPA (2013) for more complete descriptions and rules.

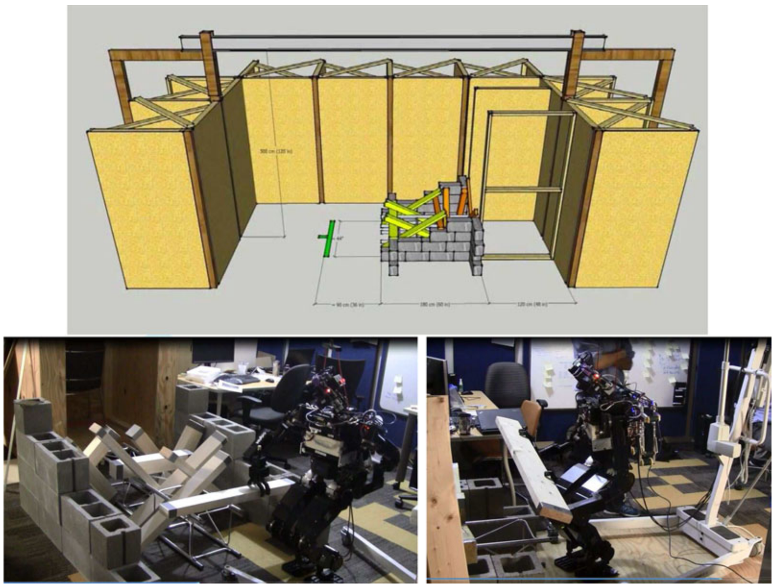


Figure 8. Debris removal event. Top: event schematic, from DARPA (2013). Bottom left: Testing at GA Tech. Bottom right: Two-armed grasping for heavier pieces was implemented early on, but became unnecessary as the task descriptions evolved.

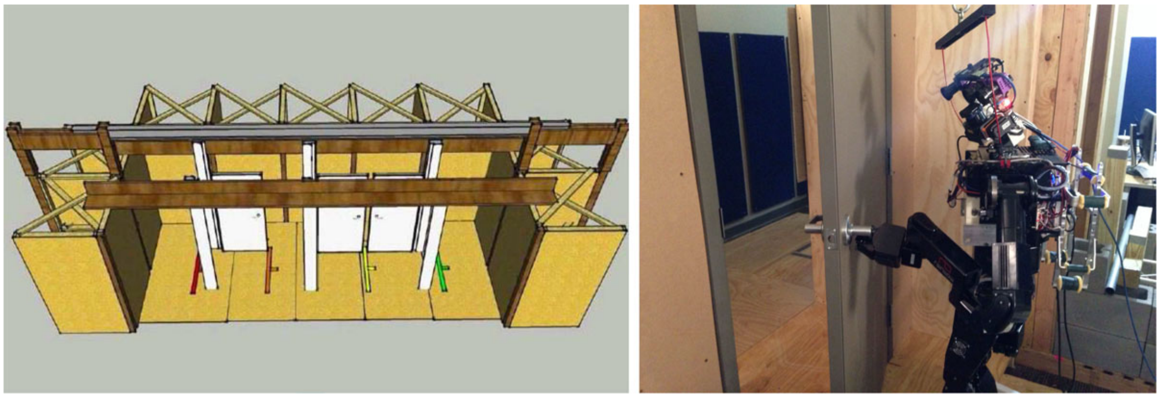


Figure 9. Door opening event. Left: event schematic, from DARPA (2013). Right: testing at GA Tech.

4.1. Debris Removal

In this event (see Figure 8), the robot is required to remove ten pieces of lightweight lumber between two walls in front of a doorway, and subsequently walk through the door. Scoring is based upon completion of three subtasks: removing the first five pieces of debris, removing an additional five, and walking through the doorway.

Shortly after the DRC-HUBO hardware arrived in the lab in August, our system could barely clear a single piece of debris within the time limit; however, as the software matured and the operators gained experience, the system could clear five debris pieces within the time limit, both in the lab at GA Tech, and at the dry-run at Drexel. During our final practice for debris clearing on the morning of the trial, DRC-HUBO performed as expected by clearing five debris pieces within the time limit.

4.1.1. Power Failure

The debris removal task was our subteam's first event at the DRC trials. To fit into the 15 min setup time, we performed some of our startup and calibration procedures offsite, and we maintained system power using an uninterruptible power supply (UPS) while transporting the robot to the event. At the start of the task, the UPS failed, causing the robot to fall. We were able to repair most of the damage to the robot and run without the UPS for the rest of the trials, but the failure had several ongoing consequences: we were forced to overrun our setup time for the other two

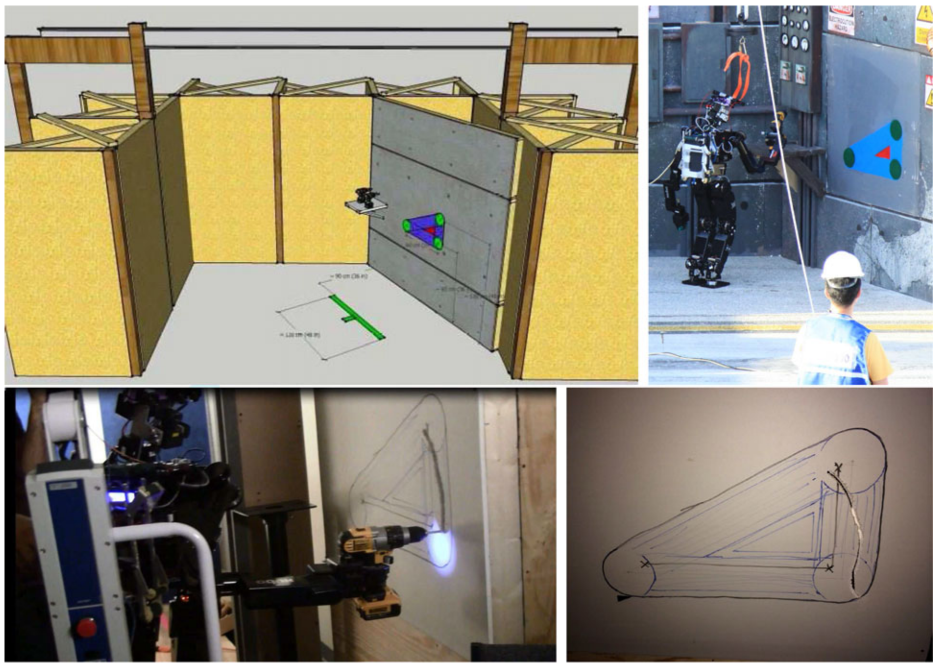


Figure 10. Wall breaking event. Top left: event schematic, from DARPA (2013). Top right: DRC trials event. Bottom: testing at GA Tech.

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 miscalibrated, impeding walking and balancing.

4.2. Door Opening

For this event (see Figure 9), 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. Points are earned for each door traversed.

In our mockup at GA Tech (see Figure 9), DRC-HUBO could reliably enter a push door, and sometimes enter a pull door within the time limit. Our strategy for push doors was to open the door slightly and subsequently push on it with a forearm while walking through the doorway. Walking speed was therefore limited by stability concerns. Our strategy for pulling was to situate the robot's feet outside of the arc of the door so the robot could open it enough to walk through without subsequently manipulating the door.

Both routine lab tests at GA Tech and the dry-run rehearsal at Drexel showed that DRC-HUBO could score at least one point by completing the first subtask. We believe that 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. We were able to approach and open the first door; unfortunately, strong wind 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 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 (see Figure 10), 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 slab of drywall. Up to three points are awarded for successfully cutting each edge of a 2 ft. by 1 ft. right triangle.

Among the events described in this section, we found wall breaking the most challenging, since the robot must use a tool to interact with and modify the environment. The task also involves locomotion while holding a heavy object. Grasping the drill was a time-consuming operation, made especially difficult by the narrow window for proper grasping (approximately ± 4 mm translation, $\pm 5^\circ$ rotation). If the robot's hand is slightly mispositioned, the trigger cannot be depressed successfully. Furthermore, drilling 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 10).

Our performance on this task varied as DARPA refined the rules and task descriptions in the months leading up to the DRC trials. 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. During a separate, brief dry-run staged by the DRC organizers in Homestead, our system successfully cut into the wall as well.

In the DRC trials, the wall breaking task was our subteam's last event. 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 again, ending the trial.

5. LESSONS LEARNED

The DRC differs in both degree and kind from most robotics research projects. Under many metrics (hours of robot time, lines of code, size of team), the DRC dwarfs the typical project experiences of the authors. Whereas a typical project might involve demonstrating a research innovation in a single domain 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. In the remainder of this section, we reflect on the lessons the DRC has taught us.

5.1. What Went Wrong

5.1.1. Event-based Task Allocation

An early strategic decision for our distributed DRC-HUBO team was to allocate each subteam's efforts by event, rather than through a systems-based approach. Consequently, there was duplication of effort across the entire team in many areas. Subteams independently developed systems for functionality that was common to various events, such as walking, user interfaces, constrained manipulation, perception, and communications. Not only is duplicated code less well tested than shared code, it is less likely to be written by the most relevant expert. Since the expertise of the subteams varied with respect to planning, control, perception, and manipulation, and each one was developing their own functionality, few software systems took full advantage of the knowledge of the entire team.

An event-centric task allocation creates perverse incentives in that time spent on releasing and maintaining shared code is time not spent on one's own event. For our team, walking and dynamic balancing (see Section 5.1.3) were the most prominent, but by no means the only casualties. It is tempting to suggest that a systems- or competency-based task allocation would have been more effective for our entire team; however, knowing *a priori* which functionality should be shared across subteams presumes global knowledge about top-level organization before the system has been designed or implemented. Faced with a future project, we believe that some combination of the two approaches would probably be best.

5.1.2. Managing Complexity

Managing the inherent complexity of robotic systems is crucial. Compared with traditional software development, dependencies between functional units are more complex, and are mediated by interaction with the physical world through an array of electromechanical systems. One illustrative example arose while debugging our walking controller, which exhibited puzzling, sporadic failures early in its development. After weeks of verifying kinematics and tweaking parameters in the walking controller, 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.

A resulting lesson is to endeavor to be as unbiased as possible when identifying root causes. Although every module was potentially a source of error, we incorrectly focused on the novel component we were developing, rather than code that preexisted it. Furthermore, it is vital 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 such controllers early on in the project, none was particularly robust, and these remained persistent weak spots for our team. In hindsight, it would have been prudent to specify a minimum set of requirements for walking and balancing as early in the project as possible, specifying resistance to various types and sizes of perturbations. Another lesson is that if no resources are explicitly allocated for common functionality in an event-based task allocation, developers may 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

Although our sensor head was adequate to the tasks required of it, both revisions to the DRC task specifications, as well as our team's shift from an initial interest in autonomy to a strong focus on human teleoperation, left room for improvements. For example, long-distance depth sensing for object detection and shape fitting became unnecessary with a human "oracle" to identify objects in monocular images. Furthermore, despite early concerns about near-distance 3D sensing for grasping ladder rungs and the roof pillars during vehicle ingress/egress, the ladder subteam chose not to use this information, and DARPA removed the ingress portion of the driving event. Human teleoperation meant compressing and downsampling data to transmit over a poor communications link, reducing the effective framerates and resolutions of the sensors. Fewer, cheaper, and smaller sensors 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 less useful for detail during fine grasping tasks.

5.1.5. Middleware Integration

For this project, we used an ROS for both the operator tools and the perception computer. While it enabled us to make rapid progress, at times our approach seemed fundamentally at odds with its design. For example, in order to prevent megabits of data from overwhelming our low-bandwidth communications link to the robot, we had to literally sever the TF tree—the distributed data structure maintaining relationships between various robot coordinate frames (Foote, 2013)—at the neck of the robot and subsequently write special-case code to stitch it back together. One lesson for us 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 that was used at the DRC trials to shape communications traffic, we simulated the DRC network conditions with a custom implementation that imposed slightly more aggressive latency and bandwidth restrictions than expected at the DRC. Upon arrival to Homestead, it became clear that the packet buffering and out-of-order delivery imposed by the DRC network imposed some surprising communications latencies, as neither one was modeled by our testing setup.

In hindsight, we aimed 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 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 the trials neared, we observed what appeared to be a substantial drop in hardware reliability. In the final two weeks, we experienced more hardware failures 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 duty cycle. It is natural for hardware usage to peak toward the end of a project, especially as software matures, 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. Counterintuitively, laboratory 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

Ultimately, we were satisfied that our operator software faithfully and effectively exposed the low-level robot functionality that we developed, and furthermore, 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. The division of labor among the operators effectively allowed each one to be reasonably active at the same time without overwhelming any operator's cognitive load. The system we produced was sufficiently general to handle the door opening, wall breaking, and debris removal events of the DRC trials. Indeed, our general-purpose approach would likely 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 code was shared across all three events, a bug fix or enhancement that benefited our performance on one could potentially aid both other events as well.

5.2.2. Simplicity of Implementations

Although other DRC teams and other subteams on our own team used sampling-based planners such as CBiRRT (Berenson, Srinivasa, Ferguson, & Kuffner, 2009) 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. Despite their utility, complete planners can add substantial complexity, by requiring programmers to explicitly model goal conditions unique to each event, as well as shapes and affordances of objects in the environment (e.g., doors, debris, drill, etc.). Several strategies we used would have been difficult to automate, especially those exploiting 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 introduces concerns about issues such as convergence and stability, kinematic singularities, joint limits, and self-collisions. Using pregenerated trajectories with statically stable starting and ending poses at zero velocity was also advantageous in preserving simplicity. 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.

5.2.3. Agile Development

Starting in October, we adopted several tenets of the agile development methodology (Larman, 2004). A whiteboard in the lab was dedicated to displaying a prioritized list of tasks. Individuals on our subteam were allocated to the highest-priority task matching their skill set, and the list was reviewed weekly. Adopting agile methods allowed us to successfully maintain the overall functionality of the system as features were added and bugs fixed. Given both the breadth of the DRC and the fast schedule, however, 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.

6. CONCLUSIONS AND FUTURE WORK

Our goal for 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 is whether we *would* in fact participate in another DRC-like project, given the opportunity. While some in the academic research community actively avoid broadly scoped, competition-based programs, in our own experience we find that these challenges stand as 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 does no good if the robot cannot reliably and robustly walk to its destination. Nevertheless, we were glad to compete at such a high level, and to work on a project that embodies the truly multidisciplinary nature of robots.

Aside from throwing into sharp relief the importance of good systems engineering practices, the DRC has also played a 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 Ph.D. program, but they also got to work with a large, integrated code base. 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. Large integrated challenges such as the DRC contribute to building a culture of competent generalists among our students.

6.1. Future work

Although we will not continue our efforts in the next phase of the DRC, we will continue to use the infrastructure we have built for this project. 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.

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, 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

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

Berenson, D., Srinivasa, S. S., Ferguson, D., & Kuffner, J. J. (2009). Manipulation planning on constraint manifolds. In Proceedings of the IEEE International Conference on Robotics and Automation (pp. 625–632). IEEE.

Buehler, M., Iagnemma, K., & Singh, S. (2007). The 2005 DARPA Grand Challenge. Springer.

Buehler, M., Iagnemma, K., & Singh, S. (2009). The DARPA urban challenge: Autonomous vehicles in city traffic. Springer.

Burke, J., & Murphy, R. R. (2004). Human-robot interactions in USAR technical search: Two heads are better than one. In Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication (ROMAN) (pp. 307–312).

Chestnutt, J. (2007). Navigation planning for legged robots. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Chestnutt, J., Lau, M., Cheng, G., Kuffner, J., Hodgins, J., & Kanade, T. (2005). Footstep planning for the Honda ASIMO humanoid. In Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona, Spain.

Dang, H., Jun, Y., Oh, P., & Allen, P. K. (2013). Planning complex physical tasks for disaster response with a humanoid robot. In Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications. IEEE.

Dantam, N., & Stilman, M. (2012). Robust and efficient communication for real-time multi-process robot software. International Conference on Humanoid Robotics (Humanoids) (pp. 316–322).

DARPA (2013). The DARPA Robotics Challenge official web site. http://www.theroboticschallenge.org/. Accessed: February 2014.

Fiser, D. (2010). libccd—Library for collision detection. http://libccd.danfis.cz/. Accessed: February 2014.

Foote, T. (2013). tf: The transform library. In Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications (pp. 1–6). IEEE.

Grey, M., Dantam, N., Lofaro, D. M., Bobick, A., Egerstedt, M., Oh, P., & Stilman, M. (2013). Multi-process control software for HUBO2 plus robot. Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications.

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

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

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., & Hirukawa, H. (2003). Biped walking pattern generation by using preview control of zero-moment point. In Proceedings of the IEEE International Conference on Robotics and Automation.

Kim, J., Park, S. W., Park, I. W., & Oh, J. H. (2002). Development of a humanoid biped walking robot platform KHR-1: Initial design and its performance evaluation. In Proceedings of the International Workshop on Humanoid and Human Friendly Robotics (pp. 14–21).

Larman, C. (2004). Agile and iterative development: A manager's guide. Addison-Wesley Professional.

Pan, J., Chitta, S., & Manocha, D. (2012). FCL: A general purpose library for collision and proximity queries. In Proceedings of the IEEE International Conference on Robotics and Automation (pp. 3859–3866). IEEE.

Phillips-Griffin, C. (2013). WPI-ARC/teleop_toolkit. Accessed: February 2014.

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

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (2009). ROS: An open-source robot operating system. In Proceedings of the ICRA Workshop on Open-Source Software.

Zhang, Y., Luo, J., Hauser, K., Ellenberg, R., Oh, P., Park, H. A., & Paldhe, M. (2013). Motion planning of ladder climbing for humanoid robots. In Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications. IEEE.

Zheng, Y., Wang, H., Li, S., Liu, Y., Orin, D., Sohn, K., Jun, Y., & Oh, P. (2013). Humanoid robots walking on grass, sands, and rocks. In Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications.