2013 13th IEEE-RAS International Conference on
Humanoid Robots (Humanoids).
October 15 - 17, 2013. Atlanta, GA

Summary of Team IHMC's Virtual Robotics Challenge Entry

Twan Koolen, Jesper Smith, Gray Thomas, Sylvain Bertrand, John Carff, Nathan Mertins, Douglas Stephen, Peter Abeles, Johannes Englsberger, Stephen McCrory, Jeff van Egmond, Maarten Griffioen, Marshall Floyd, Samantha Kobus, Nolan Manor, Sami Alsheikh, Daniel Duran, Larry Bunch, Eric Morphis, Luca Colasanto, Khai-Long Ho Hoang, Brooke Layton, Peter Neuhaus, Matthew Johnson, Jerry Pratt

***Abstract*— This paper presents a high level overview of the work done by Team IHMC (Florida Institute for Human and Machine Cognition) to win the DARPA Virtual Robotics Challenge (VRC), held June 18–20 2013. The VRC consisted of a series of three tasks (driving a vehicle, walking over varied terrain, and manipulating a fire hose), to be completed in simulation using a model of the humanoid robot Atlas. Team IHMC was able to complete all of these challenges multiple times during the competition. The paper presents our approach, as well as a bird's-eye view of the major software components and their integration.**

I. Introduction

The DARPA Robotics Challenge (DRC [1]) is a currently ongoing competition focused on developing disaster response robots that can operate in rough terrain and hazardous conditions. There are several possible routes to compete in the DRC, one of which is to first partake in the Virtual Robotics Challenge (VRC). During the VRC, which was held June 18–20 2013, teams attempted to complete three of the DRC tasks in a computer simulation using a simulated version of the robot 'Atlas', made by Boston Dynamics [2]. The three VRC tasks were to have the robot:

1) climb into a Polaris Ranger vehicle, drive along a road with obstacles, exit the vehicle and walk across finish;
2) walk across various rough terrain: a mud pit, rolling hills, and a debris field;
3) attach a hose to a spigot and subsequently turn a valve to start fluid flow.

Team IHMC completed all of these tasks multiple times in competition, and scored a total of 52 out of a possible 60 points related to task completion, placing first in a field of 22 teams.

Our preparations for the VRC include: a new whole-body control algorithm, a novel modular state estimator, a coactive-design-inspired operator user interface, and a low-bandwidth communication infrastructure. In this paper, we provide a high level overview of our approach, briefly describe each of the major software components and give insight into the way the components work together. More detailed descriptions of various parts are available in the references we cite, or will be available in future publications.

The remainder of the paper is structured as follows. The operator user interface is presented in section II. Section III provides a quick overview of our control algorithms, including our whole-body multi-contact control framework. Section IV discusses state estimation. Section V presents

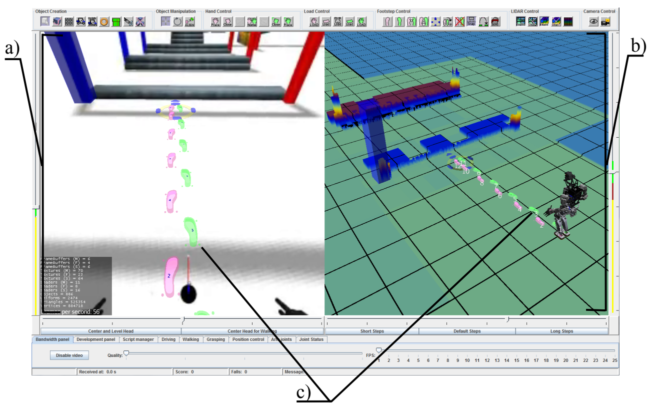

Fig. 1. Operator user interface showing: a) first-person augmented reality view; b) third-person view; and c) planned footsteps visualized in both views.

the network setup used for communication between the controller and the operator user interface. Section VI presents software development practices used to handle software complexity during the VRC. A discussion is provided in section VII. Video and further information can be found at http://robots.ihmc.us/drc/.

II. Operator User Interface

To complete the tasks of the VRC, our team relied on an operator user interface (UI) to visualize the robot in its environment, visualize the set of currently available commands, and send commands to the robot. We will first discuss the overal design (subsection II-A), followed by the main features: LIDAR data processing (subsection II-B), footstep planning (subsection II-C), the inverse kinematics-based arm positioning system (subsection II-D), and the scripting engine (subsection II-E).

A. Overall design

The driving force behind the design of our operator UI is the use of a 'coactive design' approach [3]. The basic premise of coactive design is that, in sophisticated human-machine systems, the underlying interdependence between human and machine should shape the design of the man-machine interface. The interface should allow both human and machine to contribute those features at which they excel relative to each other. In our system, the robot relies on the human operator to interpret high level information

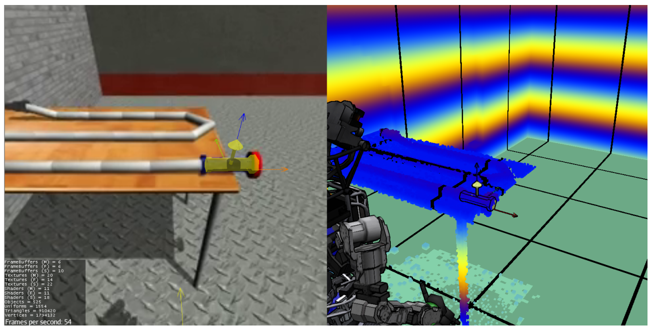
Fig. 2. First-person augmented reality camera view (left) and third-person LIDAR view (right) showing a hose on a table. The LIDAR data near the robot is higher resolution than further away. The user placed a 3D object representing the end of the hose, visible in both views, is used as the reference frame for commands and scripts when interacting with the hose.

about the world and maintain a plan of action. As the high level planner, the operator is also responsible for identifying mistakes and replanning as necessary. The operator, in turn, relies on the robot for information about the world, through high level abstractions built upon sensor data. To complete tasks in a reasonable amount of time, the operator must be able to trust these abstractions, so predictability and the minimization of automation surprise are key requirements.

As shown in Fig. 1, the UI features a first-person 'augmented reality' view in the style of [4], which overlays the robot model and operator generated objects such as footsteps and obstacles with the raw camera information. In addition, a virtual third-person view shows the same virtual objects along with a LIDAR data visualization. This third-person view is frequently more convenient for planning, and gives the operator better spatial awareness than the fixed frame that the camera allows. In this view, the operator can place and manipulate objects such as 3D models of the vehicle, valve, or spigot, which can be used as reference frames with respect to which scripted actions can be performed. We found that the ability to quickly and intuitively navigate the camera obviated the need for multiple third-person views.

We use Java Monkey Engine [5] to visualize the 3D world used in both views. In Fig. 1, a footstep list is under consideration, and can be sent to the robot by pressing the spacebar key. Showing a visualization in both views for a command such as a footstep list is fundamental to our UI, and the same mechanism is used to send hand poses, pelvis poses, torso orientations, and reference frame realignment commands. Above and below the views are control panels which also give the operator access to the commands which do not benefit from visualization such as switching controller modes, changing bandwidth settings, clearing the height map, etc. Any number of different options can be visualized before the operator commits to a plan.

B. LIDAR data processing

LIDAR data processing is used to achieve three interface objectives. First, it provides a detailed three dimensional

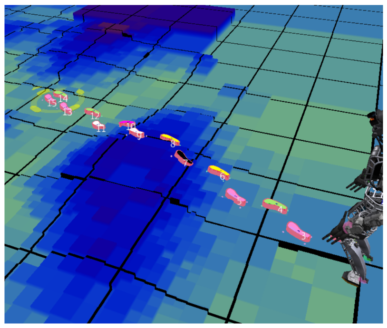
Fig. 3. Footsteps use the height map to find normals on rolling terrain. The dark footstep alerts the user to a precarious steep upslope.

representation of nearby objects, so that the operator can locate these objects in space in order to avoid collisions and to setup reference frames for higher level scripts. Second, LIDAR provides a height map of the ground which the footstep planner uses to compute footstep height and orientation. Third, since the VRC competition limits bandwidth use, our LIDAR processor must compress the data sent over the low bandwidth downlink channel. To achieve these goals, we use a quadtree [6] to represent the lowest LIDAR points for any horizontal location and a variable resolution octree [7] for any points that lie above the quadtree. The software maintains two copies of this dual tree structure: one on the field computer and one on the team computer. The bandwidth objective is achieved by only sending down those points whose insertion caused the structure on the field computer to change state. Fig. 2 shows both the octree and the quadtree together, as well as a 3D model of a hose, which the user has lined up using the octree and camera visuals.

C. Footstep planning

The interface strives to provide abstraction when it benefits the user, but retains the ability to work at a lower level in case the abstraction fails. This is the impetus behind the footstep planning system. At the top level, the operator can specify a point in either the first-person or third-person view towards which to robot should walk, causing the controller to autonomously plan and execute its own footsteps. We call this mode "blind walking", since the robot uses no remote sensor information and assumes the ground is planar. This mode is either used when the ground is known to be planar, or when LIDAR data are known to be inaccurate, such as when wading through a mud pit. At a lower level of abstraction, the operator has the option to plan footsteps on the UI side. In this mode, clicking in the UI creates a series of footsteps generated using ground height information from the LIDAR quadtree. Fig. 3 demonstrates how footsteps in a path are adjusted to match the slope and height of the ground below using the quadtree. Various footstep colors notify the user of dangerous tilt levels, which can cause the robot to reach its ankle joint limits. In cases like this the user can descend down another level of abstraction, individually

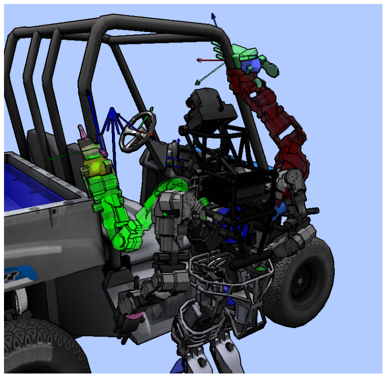

Fig. 4. Inverse kinematic arm graphics visually distinguish a feasible left hand pose (light / green) and an implausible right hand pose (dark / red).

adjusting any bad footsteps by dragging them around.

D. Inverse kinematics

When trying to grab or manipulate objects, it is often difficult for the operator to know the workspace of a manipulator, particularly when the manipulator has a reduced workspace compared to the human operator's arm. To help give the operator awareness of the workspace when finding a hand pose, we implemented an inverse kinematics visualizer, shown in Fig. 4. When the operator moves a graphic representation of a desired hand pose around, a semi-transparent image of the resulting arm configuration is shown. If the hand pose is kinematically reachable, the arm is drawn in green. Otherwise the closest possible pose is drawn in red. This visualization tool is especially useful when generating vehicle ingress and egress scripts. The lack of hard realtime requirements and robustness requirements on the UI side warrants the use of a simple, inefficient numerical inverse kinematics method, using a combination of gradient descent and randomized search. A dynamic simulation preview of the planned motion is an area of future work.

E. Scripting engine

Perhaps the highest abstraction level feature of the UI is its support for recording and executing a sequence of robot commands as a modifiable script. Once the record button is pressed, the network processor serializes all communication packets received from the UI into an XML script file. Any pose commands are transformed to a user-specified reference frame before serialization. This feature allows e.g. playback of a vehicle ingress script relative to the user positioned vehicle graphic and thus allows identical ingress behavior across multiple runs with differing vehicle poses. For the VRC we relied on executing prerecorded scripts for vehicle ingress and egress and for fastening the hose to the spigot and tightening the hose.

To keep our approach more general and to reduce automation surprises, instead of purely playing back scripted commands, each command in a script shows up in the user interface as if the operator just created the command. This allows the operator to see what the robot is going to do next and allows for online adjustments to the command if necessary to make it appropriate for the current state of the robot and world. The operator can also choose to skip individual commands, or to step backward through a script in order to repeat previous commands. This greatly improves robustness relative to pure script playback.

These user interface features together provide a stable, efficient, and trustworthy UI for controlling a humanoid robot and make effective use of the human-robot team.

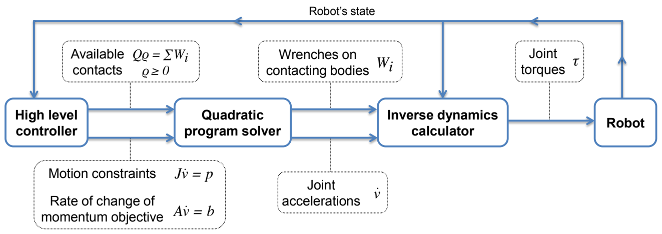
Robot's state
Available contacts $Q\varrho = \Sigma W_i$ $\varrho \geq 0$
Wrenches on contacting bodies W_i
Joint torques τ
High level controller
Quadratic program solver
Inverse dynamics calculator
Robot
Motion constraints $J\dot{v} = p$
Rate of change of momentum objective $A\dot{v} = b$
Joint accelerations $\dot{v}$

Fig. 5. Overview of information flow in the lower level parts of the controller

III. Controls

This section presents the control algorithms used to perform the VRC tasks. We primarily use a novel whole-body control framework (subsection III-A), with a number of high level behaviors built on top of it (subsection III-B). In case of emergencies, e.g. to get up after a fall, we use a library of joint-space trajectories (subsection III-C) combined with high-gain joint PD control.

A. Whole-body control framework

We designed a new whole-body motion control framework for the VRC. Fig. 5 shows the flow of information in this framework. It is heavily influenced by recent work from Lee and Goswami [8], and also incorporates features from work by Sentis [9].

A high level behavior, such as a walking or driving behavior, supplies the whole-body control framework with the following data: 1) motion constraints, 2) a rate of change of centroidal momentum objective, and 3) available contact information.

A motion constraint with index i can be written as:

$$J_i \dot{v} = p_i$$

where $\dot{v} \in \mathbb{R}^{n+6}$ is the vector of joint accelerations, including the spatial acceleration of the pelvis with respect to the world. Here, n is the number of actuated degrees of freedom. We mainly use three types of motion constraints:

1) joint space acceleration constraints, in which case $J_i \in \mathbb{R}^{1 \times n+6}$ is a selection matrix and p_i is the desired joint acceleration;
2) spatial acceleration constraints between rigid bodies or with respect to the world, where $J_i \in \mathbb{R}^{6 \times n+6}$ is a basic Jacobian [10] or geometric Jacobian [11] and p_i is the desired spatial acceleration minus Coriolis terms;

3) point acceleration constraints, specifying a desired acceleration of a body fixed point with respect to another body or the world, in which case $J_i \in \mathbb{R}^{3\times n+6}$ is a point velocity Jacobian and p_i is the desired linear acceleration minus Coriolis terms.

We often use motion constraints to track joint space or task space trajectories, in which case the constraint is determined using PD control with an added trajectory-based feed forward term. Load-bearing bodies are constrained to have zero spatial acceleration with respect to the world.

The individual motion constraints, indexed 1 through k, are concatenated into a single matrix equation:

$$J\dot{v} = p \tag{1}$$

where $J = \left(J_1^T \cdots J_k^T \right)^T$, and $p = \left(p_1^T \cdots p_k^T \right)^T$.

In addition to these motion constraints, a rate of change of momentum objective can be provided in the form:

$$A\dot{v} = b \tag{2}$$

where $A \in \mathbb{R}^{6\times n+6}$ is the centroidal momentum matrix [12] and $b = \dot{h}_d - \dot{A}v$, with $\dot{h}_d \in \mathbb{R}^6$ the desired rate of change of centroidal momentum (both linear and angular). Note that the actual rate of change of centroidal momentum $\dot{h}$ is equal to the sum of all external wrenches exerted on the robot:

$$\dot{h} = A\dot{v} + \dot{A}v = \sum_{i=1}^{m} W_i + W_g \tag{3}$$

where $W_i \in \mathbb{R}^6$ is the wrench exerted at contacting body i, and W_g is the wrench due to gravity.

Contact information is supplied in the form of a contact matrix Q, built from the available points of contact (e.g. foot corner points, or points on the robot's thighs) and a set of basis vectors[1] [13]. The basis vectors form a conservative pyramid approximation of the friction cone at each contact point, so that a nonnegative linear combination of basis vectors results in a contact point force within the friction cone. For each contacting body i, we have $W_i = Q_i\rho_i$, where ρ_i is a vector of basis vector multipliers. An inequality constraint $\rho_i \geq 0$ ensures the unilaterality of the contact forces. Summing all of the contact wrenches, we obtain

$$\sum_{i=1}^{m} W_i = Q\rho, \quad \rho \geq 0 \tag{4}$$

where $Q = \left(Q_1 \cdots Q_m \right)$ and $\rho = \left(\rho_1^T \cdots \rho_m^T \right)^T$.

Combining (1), (2), (3), and (4), we arrive at the following quadratic program:

$$\begin{aligned} \text{minimize} \quad & (A\dot{v} - b)^T W_h (A\dot{v} - b) + \rho^T W_\rho \rho + \dot{v}^T W_{\dot{v}} \dot{v} \\ \text{subject to} \quad & Q\rho = \ A\dot{v} + \dot{A}v - W_g \\ & \rho \geq \ 0 \\ & J\dot{v} = \ p \end{aligned} \tag{5}$$

[1]For grasping contacts, we used a different, but similar construction. This type of contact will be discussed in a later publication.

where W_h, W_ρ, and $W_{\dot{v}}$ are weighting matrices, also determined by the high level controller. This quadratic program is solved every controller time step for the joint acceleration vector $\dot{v}$ and the basis vector multiplier vector ρ. This information is then used to calculate a vector of desired joint torques $\tau \in \mathbb{R}^n$ using a recursive Newton-Euler inverse dynamics algorithm. These desired joint torques are the commands sent from the controller to the robot. We used CVXGEN [14] to generate a custom solver.

In comparison to [8], we solve a single QP as opposed to a sequence of QPs that together approximate (5). In addition, our notion of motion constraints provides a more powerful toolbox for control, while doing PD control on $\dot{v}$ as opposed to on τ better exploits model knowledge.

B. High level Behaviors

Several high level behaviors were developed on top of the whole-body control framework. This subsection gives brief descriptions of these behaviors.

1) Direct multi-contact behavior: The most straightforward application of the whole-body control framework is the direct multi-contact behavior. This high level behavior allows the operator to directly control the main parts of the robot, namely the feet, hands, pelvis, and chest, by sending desired poses or orientations and commanding whether or not certain contact points or grasps should be used to bear the robot's weight. Cartesian space minimum jerk trajectories were used to smoothly transition between subsequent poses or orientations. PD control with added feed forward was used to determine spatial acceleration motion constraints that achieve the trajectories. This behavior was chiefly used to perform vehicle ingress and egress.

2) Walking behavior: As opposed to the direct multi-contact behavior, the walking behavior does not specify a motion constraint that controls the pelvis position. Instead, we supply the optimizer with a desired rate of change of whole-body linear momentum, using the vector $\dot{h}_d$ and the weighting matrix W_h. The desired rate of change of linear momentum is computed by a center of mass height PD controller for the vertical axis. An instantaneous capture point controller (ICP controller, [15], [16]) was used for the horizontal plane. ICP control can be used to compute a rate of change of linear momentum that is likely to be feasible. A smooth desired ICP trajectory is generated (see [17]), which corresponds to a constant centroidal moment pivot (CMP, [18]) at the stance foot center during single support and a smooth transition of the CMP from previous to upcoming foot center during double support. This avoids jumps in the desired ground reaction forces, while support polygon constraints are likely to be fulfilled. We consider the upcoming 3 footstep locations in the design of the desired ICP trajectories.

3) Driving behavior: Due to a contact instability bug in the simulator when the robot is seated, we chose to have the pelvis hover over the seat without contacting it, using only the left hand and left foot for balance and support. The right hand is used for steering: the driving behavior specifies a point acceleration motion constraint, which controls a certain hand-fixed point to move along an arc while holding the steering wheel, resulting in steering wheel angle control. No orientation control is done for the steering hand, since arm joint limits and kinematics preclude the achievement of arbitrary hand orientations while steering. The right foot is used to press the gas and brake pedals. This is achieved using a point acceleration motion constraint for the robot's toe, and soft gain foot orientation control. Soft gains are used because arbitrary foot orientations are not always achievable due to leg joint position limits.

C. Emergencies

In parallel to the whole-body control framework, we developed a library of joint space trajectories for use in emergency cases. This trajectory library is used to recover after the whole-body control framework has failed, usually after a fall has occurred. The reason for not using the whole-body framework in this case is that it requires a good state estimate. An accurate state estimate is not available after a fall because the robot does not have sufficient sensors to know how it is touching the ground. The library includes joint space trajectory scripts for rolling over, repositioning to look around, crawling, scooting away from obstacles, emergency car egress, and standing up to transition back into the whole body control framework. Multiple ways of achieving each behavior were implemented. A Behringer BCF2000 slider board was used to design the joint space trajectories, either by directly controlling joint angles or the Cartesian locations of hands, feet, pelvis or chest using inverse kinematics. Cubic splines are used to smoothly transition between subsequent joint configurations. Gazebo's built in PD control mechanism is used to track the joint trajectories.

IV. State estimation

We wrote a novel, modular state estimator for use in the VRC. The design of this state estimator is similar to that presented in [19] and will be elaborated on in a future publication. Here, we only discuss some distinguishing features.

The goal of the state estimator is to determine the robot's position and orientation in the world, as well as its linear and angular velocity, given noisy inertial measurements. Since the DRC simulator provides perfect measurements of the robot's joint positions and velocities, the only task left to the state estimator is localization with respect to the world.

An extended Kalman filter is used to fuse the various measurements. Discretization of the process model is done online using the matrix exponential method, computed using Padé approximant methods [20]. To minimize computational requirements, special care is taken to discretize independent parts of the state model independently (resulting in smaller matrices to exponentiate), and to only discretize time-variant parts of the process model every time step. The estimator also incorporates information about the desired accelerations of the robot's links, as supplied by the whole-body control framework presented in Section III, which may give it an advantage over other floating articulated robot state estimation approaches used thus far.

The design of the state estimator is modular in the sense that it can be used with any number of inertial (angular velocity, linear acceleration, or orientation) sensors, attached to any link of the robot. In addition to inertial and joint sensors, any number of body-fixed point position and velocity measurements can be incorporated. This type of 'virtual' measurement is used when the controller has determined that certain corner points on the robot's feet are supposed to be in non-slipping contact with the stationary ground.

For the competition, the estimator ran at 1000 Hz. We only used the orientation and angular velocity measured at the pelvis link, because the linear acceleration measurement is extremely noisy and sensor information from the inertial sensors in the head and hands is not published synchronously with other sensor information. The performance of the state estimator is more than adequate for the VRC, as demonstrated by only slightly reduced controller performance compared to having perfect global state information, and the fact that we were able to build up accurate LIDAR data maps of entire VRC task worlds without special LIDAR data association techniques. Quantitative measures of performance will be supplied in a more technical follow-up paper.

V. Network setup

This section describes how the main network components required to control the simulated robot are linked together. The network layout (subsection V-A) reflects whether components have real-time requirements or not. Bandwidth between the operator user interface and the control software was artificially limited during competition to more closely simulate the physical robot's usage in the field. This required us to design a custom communication protocol (subsection V-B), and use data compression (subsection V-C).

A. Network layout

The network layout is depicted in Fig. 6. For the VRC competition, OSRF and DARPA provided the DRC simulator (Gazebo) to simulate Atlas and the environment. During the competition, the DRC simulator ran on a DARPA controlled cloud computer.

Our control software communicates with the DRC simulator over a ROS layer that allows synchronization between the simulator and controller. The DRC simulator provides sensor data updates at 1000 Hz, while our control software provides joint torque commands at 200 Hz.

Processing of remote sensor data, i.e. LIDAR data and camera images, is done on the network processor. Separating the low level control from remote sensor data processing avoids potential delays due to garbage collection and usage of all resources. The network processor is also responsible for the filtering, compression, and decompression of the data that is sent to the Operator Control Unit (OCU). The network processor sends the OCU the left camera image and the camera pose with respect to the world, the raw LIDAR scan

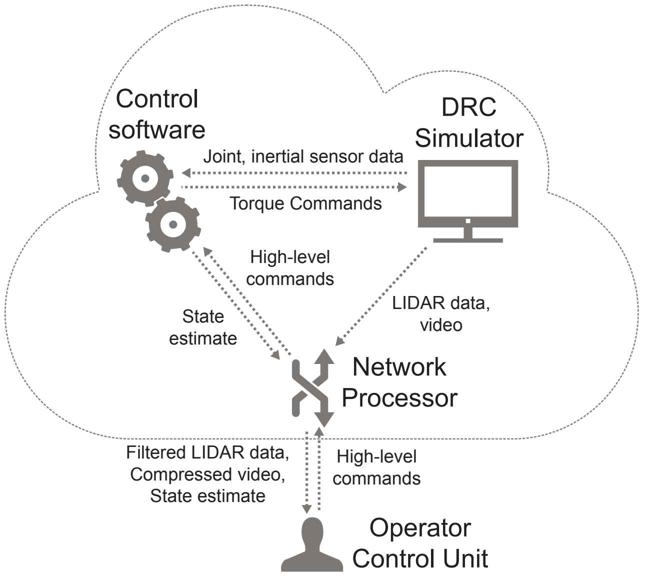

Fig. 6. Network layout for the IHMC VRC entry. The control software communicates with the DRC simulator through a high bandwidth connection at a rate of 1000 Hz. The Operator Control Unit communicates through a network processor at low bandwidth and high latency.

ranges and the current LIDAR sensor pose with respect to the world, as well as joint angles and the pose of the pelvis with respect to the world. This allows us to recreate the 3D environment, to be displayed to the user.

The main data types sent from the OCU to the network processor are lists of footsteps, finger configuration commands (open hand, close hand, point finger, etc.), desired poses and orientations of certain body parts (e.g. hands, pelvis, chest, head, feet), and load bearing commands. Moreover, we are able to send control commands to the network processor to enable/disable LIDAR and video as well as abort the current footstep list. A complete list of commands is beyond the scope of this paper.

B. Communication protocol

In the final VRC competition, the bandwidth limit was varied between runs. The most limited bandwidth restriction consisted of an average of 64 bits/s or 115,200 bits total from the OCU to the network processor and 32 kbit/s or 58,982,400 bits total from the network processor to the OCU. The actual bandwidth is limited only by the internet connection between the OCU and the network processor, but the connection was cut off upon reaching the total bit limit. The overhead of the TCP protocol was not counted towards the bandwidth limitation, which allowed us to assume that the link is reliable and all packets arrive at the destination intact and in the correct order.

We designed a simple, custom message protocol to meet these requirements. Each message consists of a header, an optional packet length and a data section, (see Fig. 7).

C. Data compression

This subsection describes data compression techniques used to achieve the limited bandwidth requirements. With

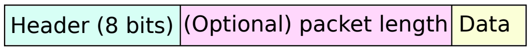

Fig. 7. Custom message protocol. Each message consists of an 8 bit header, an optional packet length and a data section.

these compression techniques we were able to achieve an uplink rate of approximately 10 to 20 bits per second and a downlink rate of approximately 20 to 40 kbit/s for the VRC challenges.

1) LIDAR data compression: The simulated Hokuyo LIDAR performs 720 scans per sweep at 40 sweeps per second. Only a fraction of these scans will change the 3D world displayed to the user (see Section II). The network processor keeps track of the 3D world, and if a scan changes the world, the scan index and range is appended to the LIDAR message. Each sweep also contains the pose of the LIDAR sensor relative to the world at the time of the scan. The resulting bandwidth used by the LIDAR data is approximately between 10 kbit/s and 40 kbit/s, depending on the complexity of the terrain and the motion of the robot. During the VRC competition we relied heavily on LIDAR for situation awareness, localizing the robot with respect to the car, determining good footsteps over the terrain, and determining the location of the hose, spigot, and valve.

2) Video compression: The left camera image is compressed using the H.264 codec. Full image I-frames are limited to 1 per 100 frames. The quality is user selectable in the UI, from 240×240 at lowest quality to 600×600 at medium quality. To further reduce the bandwidth, the number of frames per second (FPS) is user selectable from 1 to 10. At 1 FPS and the lowest quality setting, the video data takes approximately 10 kbit/s. Changes in quality require sending a new I-frame and are expensive, and should be avoided by the operator. During the VRC competition we found that video was most useful for the human to obtain a general sense of awareness of the environment, verify the state of the various controls of the vehicle, and help in aligning the hose to the spigot. Typically we found that low resolution, low frame rate, video was adequate for those purposes.

3) Pose data compression: The pelvis pose in world frame, stamped with an 8 bit identifier (ID), is sent to the OCU at 5 Hz. A buffer of sent pelvis poses and the corresponding IDs is kept at the network processor side. Footstep lists and hand and foot poses are sent relative to the latest received pelvis pose, denoted by the corresponding pelvis pose ID. The network processor can then reconstruct the desired pose in world. The maximum distance between desired foot or hand poses is bounded, allowing the use of a fixed point representation of the desired pose.

Poses are encoded as a 3D vector for the Cartesian position and a quaternion for the orientation. Depending on the desired resolution, the position elements are between 8 and 15 bits and the orientation elements between 6 and 12 bits. At the lowest resolution, this results in 64 bits per pose (8 bit header, 8 bit pelvis pose ID and 48 bit pose data). It is our experience that the operator will send desired poses at a rate much lower than 1 per second.

Footstep lists are unique in the respect that each new footstep is relative to the previous footstep and only the first footstep is relative to the pelvis. Care is taken to avoid growing errors by using the difference between the reduced resolution pose of the previous footstep and the current footstep.

VI. Software Development Practices

Software used to control humanoid robots that operate in unstructured natural environments for real-world tasks is inherently complex. Managing this complexity during development is a major issue that needs to be handled in a systematic way. Over the past decade or so we have been improving our software development tools and practices, and have been heavily influenced by the Agile Programming community [21]. Our software practices and tools were critical for us during the VRC and allowed new bugs to be detected quickly. During the final weeks before the challenge, 22 developers (one third being less experienced interns) were continuously modifying the entire tightly coupled code base, with relatively few incidents of the code base being broken. Through use of the practices described below, we were able to make critical changes within days with confidence.

A. Object Oriented Programming

All of Team IHMC's VRC software is developed in Java using standard Java Object Oriented Programming practices. Contrary to common misconceptions, Java is a fast programming language that is well suited for real time control of robots. While various vendors provide real time versions of Java, for the VRC we found that the standard version of Java was reliably able to achieve loop rates of 1 ms for our state estimator and 5 ms for our control algorithm. The only special consideration we had to make was to preallocate memory, since standard Java garbage collection can starve other threads for several milliseconds at a time. We used the YourKit Java profiler to profile memory usage and reduced runtime allocation so that missed control ticks were virtually unnoticeable. On our real robots we use a real time variant of Java which includes a preemptible garbage collector.

We find that programming is partly an act of communicating across time with another individual (often yourself) who will someday later read your code. Therefore, we take care to write fairly clean and readable code, and have adopted practices such as using descriptive names for classes, methods, and fields; reducing code duplication; favoring obvious method names over comments; and striving to have each method do a small amount of work [22].

B. Obsession with Testing

When a large number of developers are simultaneously working on a large code base it is easy to inadvertently introduce bugs, which can take significant time to fix. In addition, a high occurrence of bugs encourages a team to hold back on changes and limit the degree to which junior members of the team are trusted to modify the code base. Therefore, we try to write automatic unit tests for all of our software and often have the attitude that software can only be trusted to do what its unit tests demonstrate that it can do.

For some software modules that are easily testable, we occasionally use the practice of Test Driven Development [23] in which unit tests are written before the actual code is written. In addition to small unit tests that run on the order of fractions of a second per test, we also write end-to-end tests that can take minutes each to run. As an example, we have a sequence of tests of walking over various terrain. These tests are run nightly and automatically generate videos so that each morning we can view the videos to make sure no major breaks have occurred in the code on the previous day.

C. Continuous Integration

Rather than integrate large code modules near the end of a project, we strive to take a "full slice of the cake" [24], first developing a low-feature end-to-end version of the software and then continuously adding to that scaffolding. We typically do a small scale, often informal, design up front, and then refine the design through continuous improvements, testing, and refactoring, until an adequate design emerges through this process. We continuously integrate our developments and encourage people to check in code several times per day into our Subversion code repository. Each time a check in occurs, our Atlassian Bamboo server automatically pulls the code and runs our suite of test cases on it. If any tests fail, the server emails those responsible for any new commits since the last test ran.

D. Simulation and Visualization Tools

For the VRC, we used the Yobotics Simulation Construction Set (SCS), developed at IHMC, and Gazebo [25], by the Open Source Robotics Foundation. Both of these simulation tools have a rigid body simulation engine, contact models, and various graphical user interfaces. While the VRC required the use of Gazebo, we continued to also develop in SCS since we have access to the code base, it runs on Windows, Mac, and Linux, and it has some features that we find critical for quickly developing algorithms, such as the ability to rewind a simulation, change control parameters, and restart simulating from that point. In addition, SCS allows for easy addition of visualization graphics. For any software module that can be visualized, we typically create a visualization tool, so that the steps of an algorithm, as well as its results can be verified and better understood by a human. Simple examples include drawing reference frames, the path of Cartesian trajectories, support polygons, center of mass and center of pressure locations, etc. These graphical objects are superimposed in the simulated environment, are saved in the data file of a simulation and can be played back, and can be included in movie generation. We frequently test our software using a custom Gazebo plugin known as the Gazebo State Communicator which allows us to combine Gazebo's contact models and physics with SCS's visualizations, controller access, and rewindability.

VII. Discussion

The DARPA Robotics Challenge provides a unifying set of challenges toward the goal of having robots traverse difficult terrain and perform useful work in real-world situations. While the VRC only took place in a simulation environment, we are confident that many of the tools, techniques, and software modules that were developed will be directly applicable to the real robot in the next stage of the DRC. Areas that need further development before the physical competition include motion planning with collision avoidance, robustness to model uncertainty, and further high level behavior development. We are also evaluating better user input devices.

Fundamental to the VRC is creating a highly reliable robotic system. High reliability is essential to many real-world problems, such as disaster situations, where it is not practical or possible for a human to intervene. Therefore, a major focus of our approach was on reliability, from a high level operational view down to reliability of each software module. We emphasized automatic unit tests of our software, as well as extensive end-to-end manual testing and operator practice. For each subtask we strived to find multiple ways to accomplish the task so that there was a fallback when the preferred way did not work.

Providing an operator with an intuitive user interface is critical to remotely operating robots. We found that operators are most competent when they know what the robot is doing, why it is doing so, what it will do next, and how the operator can modify the robot's actions. By using a coactive design approach we seemed to have very few automation surprises and little operator frustration. There is a long way to go from the limited set of tasks required for the VRC to general remote operation of humanoid robots, but we are confident that our coactive design approach will help us push forward on a wide variety of tasks.

We believe that humanoid robots will someday soon be useful in real world disaster response scenarios. Control techniques such as the presented whole-body control framework provide a systematic way to control a humanoid robot operating in complex environments and manipulating various objects and tools. Thoughtful user interface design allows successful remote operation, even with low bandwidth and high latency communication. Recent improvements to sensor technologies like LIDAR provide the information required for detecting objects, localizing the robot, and performing accurate foot placement or grasps. Finally, recent robot designs are showing that it is possible to achieve the necessary kinematic, energetic, computational, and sensory requirements in a humanoid robotic platform.

However, as the long author list on this paper suggests, developing software for remote operation of humanoid robots requires a large team of dedicated developers with a large range of skills and interests, as well as many resources. We are happy to have been part of the DARPA Virtual Robotics Challenge and are looking forward to what lies ahead.

References

[1] Defense Advanced Research Projects Agency. (2013) DARPA DRC | DARPA Robotics Challenge Home. www.theroboticschallenge.org.

[2] G. Nelson, A. Saunders, N. Neville, B. Swilling, J. Bondaryk, D. Billings, C. Lee, R. Playter, and M. Raibert, "PETMAN: A Humanoid Robot for Testing Chemical Protective Clothing," *Journal of the Robotics Society of Japan*, vol. 30, no. 4, pp. 372–377, 2012.

[3] M. Johnson, J. Bradshaw, P. Feltovich, C. Jonker, B. Riemsdijk, and M. Sierhuis, "The Fundamental Principle of Coactive Design: Interdependence Must Shape Autonomy," in *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, ser. Lecture Notes in Computer Science, M. Vos, N. Fornara, J. V. Pitt, and G. Vouros, Eds. Springer Berlin Heidelberg, 2011, vol. 6541, pp. 172–191.

[4] J. Carff, M. Johnson, E. M. El-Sheikh, and J. E. Pratt, "Human-Robot Team Navigation in Visually Complex Environments," in *IROS*, 2009.

[5] M. Powell. (2013) JMonkey Engine. www.jmonkeyengine.com. Maintained by open source community.

[6] G. Sidhu and R. Boute, "Property Encoding: Application in Binary Picture Encoding and Boundary Following," *IEEE Transactions on Computers*, vol. C-21, no. 11, pp. 1206–1216, Nov. 1972.

[7] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129–147, Jun. 1982.

[8] S.-H. Lee and A. Goswami, "A Momentum-based Balance Controller for Humanoid Robots on Non-level and Non- stationary Ground," *Autonomous Robots*, vol. 33, no. 4, pp. 399–414, 2012.

[9] L. Sentis, J. Park, and O. Khatib, "Compliant Control of Multicontact and Center-of-Mass Behaviors in Humanoid Robots," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 483–501, Jun. 2010.

[10] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, Feb. 1987.

[11] V. Duindam, "Port-Based Modeling and Control for Efficient Bipedal Walking Robots," Ph.D. dissertation, University of Twente, 2006.

[12] D. E. Orin and A. Goswami, "Centroidal Momentum Matrix of a humanoid robot: Structure and properties," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2008, pp. 653–659.

[13] N. Pollard and P. Reitsma, "Animation of Humanlike Characters: Dynamic Motion Filtering with a Physically Plausible Contact Model," in *Yale Workshop on Adaptive and Learning Systems*, 2001.

[14] J. Mattingley and S. Boyd. (2013) CVXGEN: Code Generation for Convex Optimization. http://cvxgen.com/.

[15] T. Koolen, T. de Boer, J. Rebula, a. Goswami, and J. Pratt, "Capturability-based analysis and control of legged locomotion, Part 1: Theory and application to three simple gait models," *Int. J. Robotics Research*, vol. 31, no. 9, pp. 1094–1113, Jul. 2012.

[16] J. Englsberger, C. Ott, M. A. Roa, A. Albu-Schaffer, and G. Hirzinger, "Bipedal walking control based on Capture Point dynamics," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sep. 2011, pp. 4420–4427.

[17] J. Englsberger and C. Ott, "Integration of vertical COM motion and angular momentum in an extended Capture Point tracking controller for bipedal walking," in *IEEE-RAS International Conference on Humanoid Robots*, Osaka, Japan, 2012.

[18] M. B. Popovic, A. Goswami, and H. Herr, "Ground Reference Points in Legged Locomotion: Definitions, Biological Trajectories and Control Implications," *Int. J. Robotics Research*, vol. 24, no. 12, pp. 1013–1032, 2005.

[19] M. Bloesch, M. Hutter, M. Hoepflinger, C. D. Remy, C. Gehring, and R. Siegwart, "State Estimation for Legged Robots - Consistent Fusion of Leg Kinematics and IMU," in *Robotics: Science and Systems VIII*, Sydney, NSW, Australia, 2012.

[20] C. Van Loan, "Computing integrals involving the matrix exponential," *IEEE Transactions on Automatic Control*, vol. 23, no. 3, pp. 395–404, Jun. 1978.

[21] K. Beck, "Extreme Programming Explained," *Embrace change*, 2006.

[22] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Prentice Hall, 2008.

[23] K. Beck, *Test driven development: By example*. Addison-Wesley Professional, 2003.

[24] J. Rasmusson and S. D. Pfalzer, *The Agile Samurai: How Agile Masters Deliver Great Software*. Pragmatic Bookshelf, 2010.

[25] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. 2004 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, vol. 3. IEEE, 2004, pp. 2149–2154.