Valkyrie: NASA's First Bipedal Humanoid Robot

Nicolaus A. Radford, Philip Strawser, Kimberly Hambuchen, Joshua S. Mehling, William K. Verdeyen, Stuart Donnan, James Holley, Jairo Sanchez, Vienny Nguyen, Lyndon Bridgwater, Reginald Berka, Robert Ambrose[*]
NASA Johnson Space Center

Christopher McQuin
NASA Jet Propulsion Lab

John D. Yamokoski
Institute of Human Machine Cognition

Stephen Hart, Raymond Guo
General Motors

Adam Parsons, Brian Wightman, Paul Dinh, Barrett Ames, Charles Blakely, Courtney Edmonson, Brett Sommers, Rochelle Rea, Chad Tobler, Heather Bibby
Oceaneering Space Systems

Brice Howard, Lei Nui, Andrew Lee, Michael Conover, Lily Truong
Jacobs Engineering

David Chesney
Wyle Laboratories

Robert Platt Jr.
Northeastern University

Gwendolyn Johnson, Chien-Liang Fok, Nicholas Paine, Luis Sentis
University of Texas Austin

Eric Cousineau, Ryan Sinnet, Jordan Lack, Matthew Powell, Benjamin Morris, Aaron Ames
Texas A&M University

[*]Due to the large number of contributors toward this work, email addresses and physical addresses have been omitted. Please contact Kris Verdeyen from NASA-JSC at william.k.verdeyen@nasa.gov with any inquiries.

Abstract

In December 2013, sixteen teams from around the world gathered at Homestead Speedway near Miami, FL to participate in the DARPA Robotics Challenge (DRC) Trials, an aggressive robotics competition, partly inspired by the aftermath of the Fukushima Daiichi reactor incident. While the focus of the DRC Trials is to advance robotics for use in austere and inhospitable environments, the objectives of the DRC are to progress the areas of supervised autonomy and mobile manipulation for everyday robotics. NASA's Johnson Space Center led a team comprised of numerous partners to develop Valkyrie, NASA's first bipedal humanoid robot. Valkyrie is a 44 degree-of-freedom, series elastic actuator-based robot that draws upon over 18 years of humanoid robotics design heritage. Valkyrie's application intent is aimed at not only responding to events like Fukushima, but also advancing human spaceflight endeavors in extraterrestrial planetary settings. This paper presents a brief system overview, detailing Valkyrie's mechatronic subsystems, followed by a summarization of the inverse kinematics-based walking algorithm employed at the Trials. Next, the software and control architectures are highlighted along with a description of the operator interface tools. Finally, some closing remarks are given about the competition and a vision of future work is provided.

1 Introduction

The DARPA Robotics Challenge is a worldwide competition aimed at advancing the state-of-the-art in robotics with respect to mobile manipulation and supervisory control. The DRC was inspired in part by the consequences of the Fukushima Daiichi reactor incident which illustrated, rather candidly, how inadequate current robotic technologies are for use in highly unstructured human environments. To help remedy this shortcoming, DARPA organized an industrial disaster challenge consisting of eight representative tasks required of a robot. A sampling of the tasks includes climbing ladders, handling debris, turning valves, and mating hose connectors. The tasks are similar to those required of a robotic astronaut assistant in an extraterrestrial setting, like Mars or the moon. NASA sees considerable overlap in the robotic technologies being developed for the DRC and those needed to advance human spaceflight beyond low earth orbit. Therefore, NASA adopted a long range approach and developed Valkyrie with the ultimate goal of creating a robotic platform that is capable and effective in both space and Earth-bound applications.

NASA's Johnson Space Center led a team of external partners, both academic and industrial, to develop NASA's first bipedal humanoid robot. Valkyrie is the culmination of almost two decades of humanoid robotics research that has focused on manipulation, dexterity, wheeled mobility, and wearable robotic devices (Ambrose et al., 2000; Diftler et al., 2011; Mehling et al., 2007; Harrison et al., 2008; Rea et al., 2013). Even so, Valkyrie was a "clean sheet" design that was conceived, designed, developed, manufactured, assembled and verified in less than 12 months. The Valkyrie effort represents numerous advancements in robotics in the areas of rotary series elastic joints, embedded motion control, energy storage, embedded computing, distributed control, pressure-based tactile sensing, supervisory control, operator interface design and electric motor research.

2 Mechatronic Design

2.1 Mechatronic Overview

Valkyrie stands 1.87m tall, weighs 129kg, and approximates a human range of motion. Drawing heavily from lessons learned from past NASA robots (Lai-Fook and Ambrose, 1997; Lovchik and Diftler, 1999; Ambrose et al., 2000; Diftler et al., 2003; Bridgwater et al., 2012), the hardware design team was able to vastly improve performance of the robot while at the same time reduce weight. While the robot's main design requirements were chosen to ensure the ability to compete in the DRC tasks, NASA's involvement required a larger view that included practical considerations beyond the scope of the DRC Trials. These include requirements for roughly human shape, size, and weight; fall protection; integrated power; and an all-electric design.

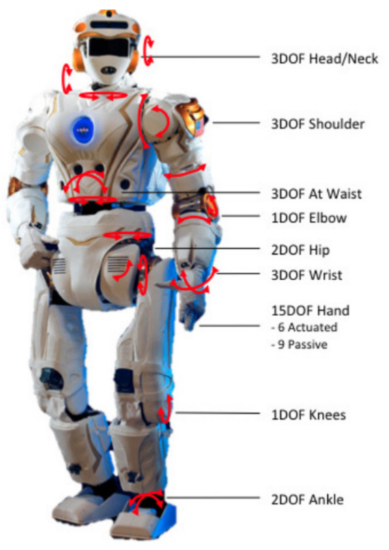
To ensure that Valkyrie could operate in human engineered environments, human ranges of motion were used to inform the workspaces of the neck, waist, arms, hands and legs. Both the robot's disaster relief mission and its space mission analogues require an ability to do repairs quickly. The DRC Trials event structure was used as a baseline for setting that repair requirement, which was a 15 minute interval between events. This drives an extremely modular design where large sections of the robot can be replaced with a single connector and one bolt. To enable this modularity, each limb was designed with integrated actuators, electrical and control systems. A layered impact protection system, consisting of hard plastic shells with energy absorbing foam and industrial vinyl covering surrounds each limb to distribute impact or fall loads to the robot hard points. These are also modular and be replaced with minimal hardware.

2.2 Limb Architecture

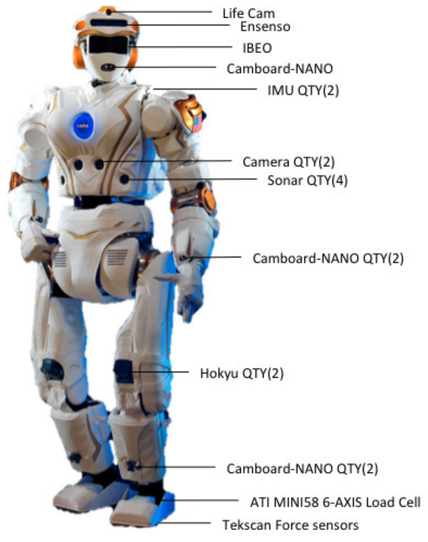
Valkyrie is a highly integrated and articulated machine with 44 actuated degrees of freedom (DOF) packaged into a humanoid form. Figure 1(a) shows the basic motions of the robot. Twenty-five of the arm, leg, and torso joints are series elastic actuators (SEA), in which a spring element is built into the joint in series with the output, based in part on previous Robonaut designs (Ihrke et al., 2012). Nine of the actuators are rigid joints and the remaining 12 actuators are tendon driven assemblies in the hands. The modular structure of the robot treats each arm, each leg, the pelvis and torso as equivalent limbs. Each limb can be removed with a single connector and one bolt, and the pelvis can be removed with a single connector and three bolts.

Each arm consists of a 3-DOF shoulder joint complex, implemented as three series elastic joints whose joint axes share a common point of intersection, along with elbow and wrist roll joints. These five primary arm joints are all implemented as rotary SEAs. A set of parallel linear actuators with single axis load cells in the two-force member completes the wrist, enabling both pitch and yaw of the hand.

Five of the tendon actuators within the forearm are the flexure actuators for the hand, an example of which is illustrated in Figure 2(a). Each hand has four under-actuated three-link fingers: a 4-DOF thumb, and three 3-DOF fingers, where the primary finger contains a passive adduction DOF. Passive elastic elements open the fingers. Additionally, a geared actuator directly drives the thumb roll. Further information concerning the design and analysis of Valkyrie's hand system can be found in (Guo et al., 2014).



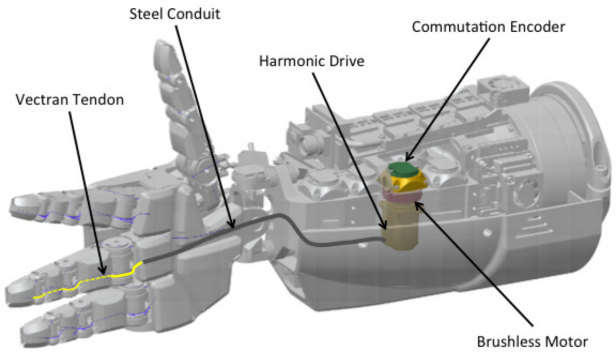
(a)



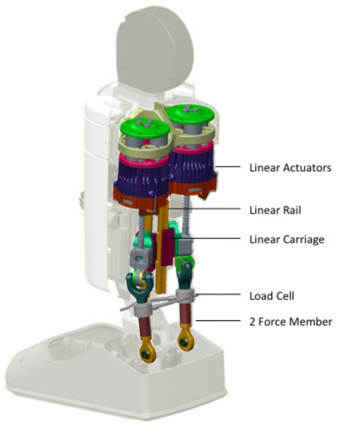
(b)

Figure 1: Valkyrie's (a) degrees of freedom and (b) Valkyrie's sensors.

The robot's 3-DOF waist consists of a pair of parallel linear SEAs providing pitch and roll on top of a rotary SEA providing yaw. Each leg is designed in a yaw-roll-pitch-pitch-pitch-roll configuration, with the first three hip joints and the knee joint implemented as rotary SEAs. Each ankle is implemented as a pair of parallel linear SEAs very similar to the waist.



(a)



(b)

Figure 2: Valkyrie's (a) finger drivetrain and (b) ankle actuators.

To minimize design effort while still packaging within the humanoid form, joints with similar performance requirements are reused across the robot. The waist linear actuators share a design with the ankle actuators. The first joint in the arms, legs and waist kinematic chain are all essentially the same actuator. The neck shares the same actuators as the wrist roll but uses traditional rigid actuators form in a pitch-roll-pitch architecture.

2.3 Actuators & Motor Control

Valkyrie's rotary joints include a frameless brushless DC motor with a high speed relative encoder on the motor rotor for commutation. Eschewing traditional Hall sensors for commutation saves motor mass and length and allows more sophisticated commutation schemes. The rotor is attached to a harmonic drive wave generator, while the stator is attached to the harmonic drive circular spline. A custom torsion spring sized for the actuator output is placed between the harmonic drive flex cup and the joint output, with a sensor mounted to the assembly that measured only the spring deflection. In most cases this was a 32-bit optical sensor. Absolute joint position was measured directly across the joint with a coaxial magnetic sensor that measured true output and joint input, bypassing the spring deflection sensor. This joint architecture eliminates the additional noise in both position and torque sensing over previous architectures that subtracted two encoder signals for differential position measurement (Diftler et al., 2011), at the cost of modest additional cable management. The desire for a more human-like form as well as the ability to gain efficiency at high speeds drove two of the leg joints, hip and knee extension and flexion, to have a belt drive between the motor and harmonic. The additional reduction in the belt drive allowed the motor to run at higher speed than the harmonic drive's input rating, while still achieving 9 rad/s at the output.

In addition to the rotary actuators, Valkyrie also contains two linear actuator designs. The wrist actuators use a prepackaged motor with a spur gear pass to drive a ball screw nut on a linear slide. These motors use hall commutation and current sensing, with similar electronics to the finger actuators. The series elastic ankle and waist linear actuators drive a roller screw nut directly, and are commutated and driven similarly to the rotary arm and leg joints. These motors are mounted on springs behind the actuator, with their linear deflection measured directly by a 32-bit sensor. Figure 2(b) illustrates the linear actuator in the ankle and a similar setup is used in the waist, differing only by their respective spring stiffnesses. All of these linear actuators feature a two-force member to allow for parallel actuation across two degrees of freedom, while magnetic absolute position encoders placed at the universal joints in the wrist, ankle and waist provide position sensing.

Advanced distributed motor control is one of Valkyrie's enabling technologies. A direct descendant of the Robonaut 2 motor controller, the same motor controller, referred here as the "Turbodriver", drives all 15 DOF in the torso, pelvis and legs as well as the first 4 DOF in each arm. At its core, the redesigned Turbodriver includes a forced-air cooled motor bridge capable of sourcing more than 30A continuously and 60A for short bursts of high torque, while both the bridge and motor windings are protected with temperature and current limits. The Turbodriver powers multiple auxiliary sensors and components and routes signals from those sensors. It also has a replaceable logic card consisting of both a dedicated microcontroller and FPGA. The motor controller can read the joint-embedded force and torque sensors and enables closed-loop control based upon on any combination of these sensors. Dual communications channels allow either redundant communication with the robot's main CPU, or a dedicated master/slave setup between parallel linear actuators. The wrist roll, neck and finger joints are driven by the Radtile, a motor controller similar to the Turbodriver, capable of sourcing up to 9A. Rather than using logic cards, Radtile boards are installed on either custom single-axis clusters (Leonidas), for the neck and wrist roll joints, or custom four-axis clusters (Athena), for the fingers and wrist linear actuators. While neither the Athena nor the Leonidas boards feature a dedicated processor, soft processors are implemented in the FPGA's to allow for reuse of some embedded code.

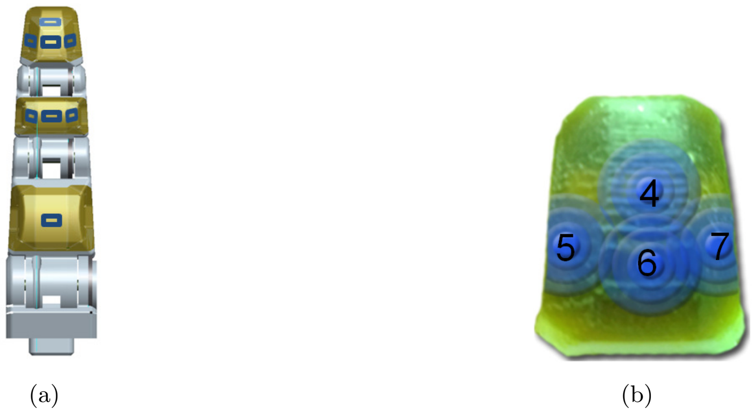


Figure 3: (a)The barometers in each finger (dark rectangles). (b)The distal barometer spatial sensitivity.

2.4 On-Board Computation & Power

Valkyrie's "brain" was implemented through three Intel Core i7 COM Express CPU's, each with 8GB of DDR3 memory. A CARMA development kit, containing a Tegra3 ARM A9 processor together with a NVIDIA Quadro 1000M GPU, allows for parallel sensor interpretation algorithms. These four computers run the high-level robot controllers and process, compress and deliver sensor data. Data can be shipped via a wired Ethernet or WiFi connection. A sophisticated power distribution and battery management board enables indefinite operation when connected to a power tether. A 14kg, dual output voltage, 1.8 kWh battery enables approximately one hour of tetherless operation.

2.5 Sensor Systems

Valkyrie is a general purpose humanoid robot, designed to operate in many human environments with the assistance of an operator. To this end, Valkyrie carries a significant sensor load, split into two main categories: proprioceptive and exteroceptive. The exteroceptive sensors can be further categorized into immediate and long-term planning sensors. These categories arose from a desire to meet the following goals: provide situational awareness to the operator, provide an actionable representation of the environment for use by the robot, provide feedback for the robot's control systems, and perform the above three goals in a bandwidth-limited and computationally constrained environment.

These goals were developed after careful consideration of the DRC Trials tasks and the possible environments in which the Trials might occur. Combining these goals with mechanical and aesthetic constraints, the result is the the sensor suite is listed in Tables 1 and 2 and shown pictorially in Figure 1(b).The sensor suite consists of: five time-of-flight (ToF) depth cameras, three LIDARS, a camera stereo pair, four USB cameras, eight load cells, two 6-axis force/torque sensors in the ankles, many pressure sensors in the hands (shown in Figure 3) and feet, two MEMS IMUs in the torso, a 9-axis INS in the head, 3-axis accelerometers and 3-axis gyroscopes in the hands and feet, and multiple encoders at each of the 44 actuated joints and Hall position sensors at the 16 underactuated finger joints.

3 Control Systems

3.1 Decentralized Embedded Control

A decentralized approach has been adopted for the control of Valkyrie's 44 degrees of freedom (Paine et al., 2014). Each individual actuator is driven by a single-joint control law running on an associated embedded motor controller. A multi-joint controller running on the robot's central computer, in turn, coordinates the efforts of each of the individual joints. This architecture has a number of benefits, particularly in light of the extensive use of series elasticity in Valkyrie's actuators. Namely, actuator-level dynamics can be abstracted away from the multi-joint level by each of the single-joint controllers. This significantly reduces the complexity of the dynamic model at the multi-joint level, easing control design while also decreasing the computational burden placed on the central control computer.

Enabled by the powerful, custom-built, embedded motor controllers integrated into Valkyrie's design, this distributed architecture also allows for highly effective, dynamic model-based control approaches - the majority of joint-related processing can be performed at 5kHz loop rates on processors colocated with each actuator. Additionally, a decentralized approach to actuator control lends itself to an incremental testing methodology in which each joint in the multi-DOF system can be tuned and validated individually, before integration into the whole. Valkyrie's SEAs can be characterized, tuned, tested or debugged individually, with little effect on the parallel development of a multi-joint controller. This is a significant benefit that eased the integration process and greatly reduced control development time during the aggressive design schedule of the DRC.

3.2 Torque Control of Series Elastic Actuators

To realize the desired hierarchical control abstraction, an embedded torque controller exists for each series elastic joint in Valkyrie such that the actuators appear, to the multi-joint controller at the high level, as ideal torque sources (or at least low pass filtered ideal torque sources, valid up to the required bandwidth). This is accomplished using the joint torque control architecture of Figure 4. Here, the physical plant is modeled as a second order system representing the SEA with a locked output (i.e. infinite output inertia), providing the following transfer function from motor current (i) to measured spring force (τ_k):

$$P(s) = \frac{\tau_k(s)}{i(s)} = \frac{N k_\tau \eta k}{j_m s^2 + b_m s + k} = \frac{\beta k}{j_m s^2 + b_m s + k} \tag{1}$$

where j_m is the motor inertia felt by the spring, b_m is the effective motor-side damping, k is the spring stiffness, N is the gear ratio of the actuator, k_τ is the motors torque constant, and η represents a measure of drivetrain efficiency. This plant is then augmented in the closed loop control of Figure 4 using a feedforward term ($N^{-1}\eta^{-1}{k_\tau}^{-1}$) and a straightforward proportional-derivative feedback controller (PD) to yield the closed loop transfer function:

$$P_c(s) = \frac{\tau_k(s)}{\tau_r(s)} = \frac{(k\beta k_d)s + k(1+\beta k_p)}{j_m s^2 + (b_m + k\beta k_d)s + k(1+\beta k_p)}. \tag{2}$$

The control gains (k_p) and (k_d) of the PD controller can be used to increase the system's bandwidth and eliminate a resonant peak in the frequency response due to the passive actuator dynamics. Thus the desired torque-source-like behavior can be achieved.

The assumption of a locked output SEA plant is, of course, not valid in the operational robot, and variation in actuator load inertia will affect the behavior of the derived closed loop system, P_c. Therefore, a disturbance observer (DOB) is also included in the joint-level controller (illustrated, again, in Figure 4). By using an inverse nominal model, ${P_c}^{-1}$, the DOB preserves and enforces a dynamic plant response in the presence of either external disturbances or, central to our application, plant model variations, like changes in output inertia. The DOB allows a control architecture built around the fixed output model of Equation 1 to generate ideal torque source behavior even when this assumption is invalidated by replacing the plant model, P, with Valkyrie's physical SEAs. A low pass filter, Q, is also included in the disturbance observer to ensure that ${P_c}^{-1}$ is proper and to provide an additional parameter to use for tuning the performance of the full control loop.

Figure 5(a) illustrates the torque tracking bandwidth of Valkyrie's series elastic joint torque controller as implemented on the robot's elbow actuator (under a fixed output constraint). Here, a significant improvement versus open loop actuator performance is observed, extending the torque bandwidth of the system from 13 Hz to 70 Hz. Additionally, the resonant peak in the open loop response is eliminated providing significant torque tracking improvement around the actuator's passive natural frequency. Extending the application of this control approach beyond a single joint with fixed output to a 4 degree-of-freedom Valkyrie upper arm interacting with its environment further demonstrates the effectiveness of the robot's joint level torque controller. In the experiment of Figure 5(b), Valkyrie's first four arm joints are commanded to track a constant torque value. A human then interacts with the robot to emulate the types of disturbances the limb might experience while performing in the field. Figure 6 is a measure of each actuator's torque tracking performance and position throughout the test. During full arm motions, the DOB-based torque control of Figure 4 not only compensates for the aforementioned variations in output inertia seen by each actuator, but it also continues to effectively track torque in the presence of disturbance torques induced on each actuator from both the environment (i.e. the human) and the other actuators in the serial chain. The peak tracking errors for the four arm joints are 0.85 Nm, 2.2 Nm, 0.63 Nm, and 0.64 Nm (J1-J4, proximal to distal). These errors represent less than 1% of the rated torque range for each actuator, thus demonstrating successful disturbance attenuation without sharing data across multiple joints. This performance enables the decentralized control approach adopted for Valkyrie.

3.3 Embedded Control Modes

While Valkyrie's joint-level torque control serves as the foundation for our decentralized approach to actuator control, the robot's embedded motor controllers also allow for additional modes that build upon and complement the aforementioned torque control. Each joint can receive a variety of inputs from the high-level, multi-joint controller depending on what operation mode is desired. As described previously, "torque control mode" receives a desired torque command and generates a response using the architecture of Figure 4. Feedback of actuator output position and velocity is provided to the multi-joint controller to generate appropriate torque commands at the whole body level. A second, closely related, "impedance control mode" receives a desired joint position and velocity, as well as a desired joint stiffness, desired joint damping, and feedforward gravity

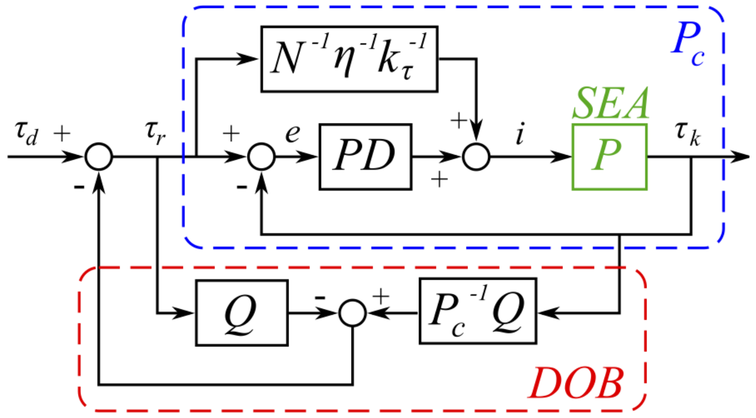
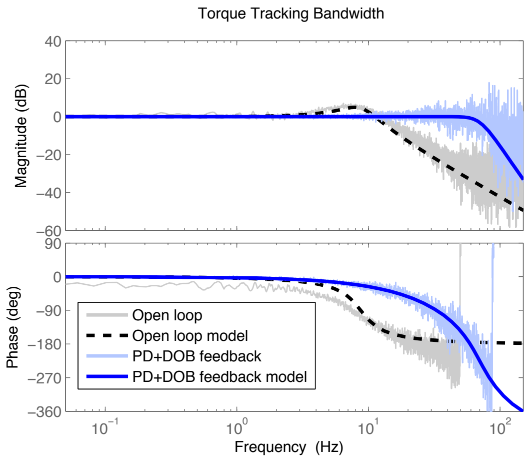
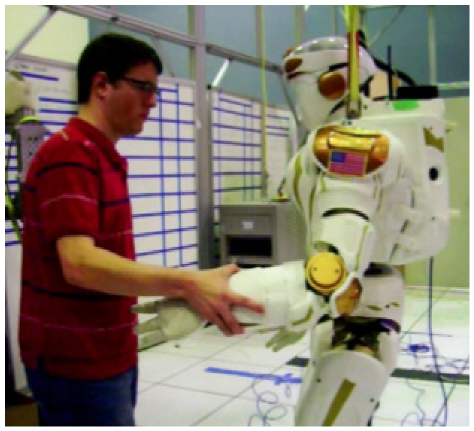


Figure 4: Diagram of Valkyrie's series elastic joint torque controller. The PD compensator is used to shape the dynamics of the torque response while the DOB ensures that the desired response is achieved in the face of external disturbances and plant model variations, like changes in output inertia.



(a)



(b)

Figure 5: (a) Frequency response of a Valkyrie elbow actuator with a fixed output. The dashed line represents the open loop response while the solid line represents closed loop performance using the controller in Figure 4. (b) Human interaction experiment with the Valkyrie left arm. The four upper arm joints are commanded to track a constant torque while the human moves the limb to impart disturbance motions on each joint.

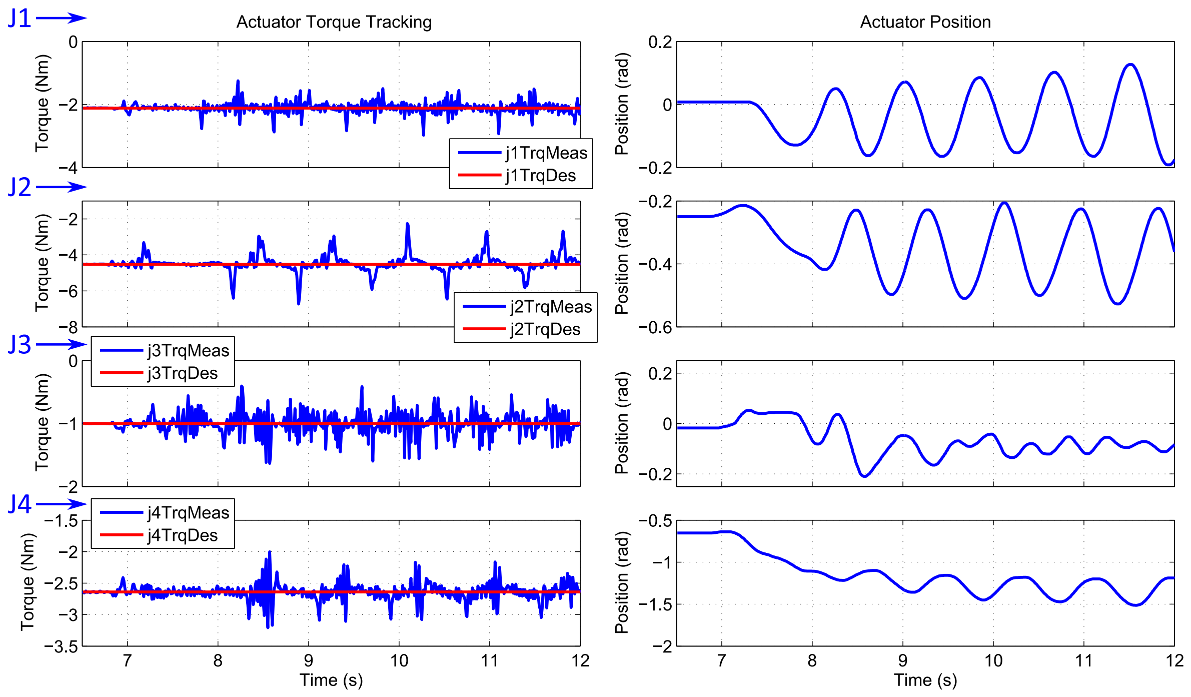


Figure 6: Torque tracking and joint position data from the human interaction experiment of Figure 5(b). Small torque tracking errors at each joint in response to human imparted motion demonstrate the applicability of Valkyrie's series elastic joint torque controller to serial chain limbs.

compensation torque. Here, an outer impedance control loop is run at the joint level, with these terms as inputs, that results in a commanded joint torque being fed to an inner torque controller. Again, joint positions can be reported back to the multi-joint controller, but the implementation of joint stiffness and damping is done locally to minimize latency and expand the effective impedance range of Valkyrie's actuators.

Each actuator also has the ability to run in a "position control mode." In this mode, as the name implies, the multi-joint controller simply sends position and velocity references to each joint and a high gain control loop drives the output of the actuator to track these reference signals. Position control mode is particularly valuable for driving Valkyrie's non-SEA joints and it also enables easy experimentation with, and quick transitions between, different high-level control schemes.

4 System Level Software Infrastructure

The system-level software for Valkyrie is distributed across multiple networked computers inside the robot. This software infrastructure is responsible for handling low-level communication and control of the robot and for providing telemetry to the supervisory control system. The three components of the system software infrastructure are the (1) **Control System**, (2), the **Bandwidth Manager**, and the (3) **Data Compression Pipeline**.

These components collectively provide a closed-loop controller framework, allow fast and efficient communication with robot hardware, and provide a compressed and streamlined method for moving data and commands between the operator and the robot.

4.1 Control System

The control system is responsible for turning task-level robot commands into low-level joint and actuator commands. This software process gathers data from the robot, processes control algorithms, and sends commands to the robot's actuators. The Valkyrie control system uses the ROS Control framework (Meeussen, 2013). This closed-loop controller framework is composed of a hardware interface layer and a controller layer. The main loop of the control system runs at a pre-defined regular interval. During each control loop, the hardware interface updates data from the robot, each active controller is executed, and the hardware interface updates commands which are sent to the actuators.

4.1.1 Hardware Interface

The primary responsibility of the hardware interface is to provide a run-time framework that bridges communication between high-level software and the robot hardware. To facilitate this framework, the subsystem provides a rich API for handling communication with robot actuators, an abstraction of low-level functionality for use with this API and tools that provide insight into the current hardware state for users. Figure 7 illustrates the interaction between the hardware interface and other levels of abstraction in the system.

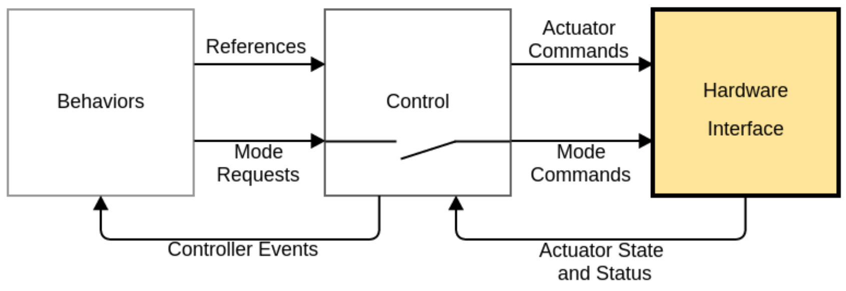


Figure 7: Interaction of the Hardware Interface subsystem with other levels in the system.

Valkyrie's distributed control architecture is composed of general purpose embedded computers and highly specialized embedded actuator controllers. While the general purpose computers communicate using standard ethernet, the embedded actuator controllers use Robonet, a custom data network. Robonet is a high-speed, two-wire, multi-point data network that contains a single master and several nodes. The master for Valkyrie connects to an embedded computer, and, in most cases, each node is a single-board computer acting as a single-axis actuator controller. Other Robonet nodes for Valkyrie include the power management system and the forearms. Because Robonet is agnostic to the data delivered, the central processor can command the power system over Robonet as easily as commanding the electro-mechanical joints. The forearm has a further hierarchical architecture: the master communicates with the forearm's central processor as a node, which then communicates with other single-axis controllers for the wrist and fingers.

The API for Robonet attempts to balance speed with flexibility. Previous attempts at APIs for

Robonet have used dynamically-generated data structures to provide fast communication between Robonet nodes. For Valkyrie, the APIBuilder tool was developed as a ROS Python module to generate both C++ and Python APIs from an XML file for Robonet nodes. The XML-based nature of this tool allows for simple creation of new API functions. The tool also creates unit tests for generated APIs. The generated C++ APIs become shared libraries that can be loaded at run-time, while the generated Python APIs are mostly used by engineering tools. The robot's hardware interface is responsible for instantiating and making hardware resources available to the controllers. Using the APIBuilder libraries to abstract details of I/O transactions, the hardware interface populates fresh data into the hardware resource data elements and sends hardware interface command elements back down to the joints.

4.1.2 Controllers

ROS Control can load many different types of controllers. These controllers include simple PD controllers for the fingers, inverse kinemeatics or Whole Body Control controllers (Sections 5.3 and 5.2) and mode controllers. While the hardware interface of the robot is fairly static, the controllers in the ROS Control framework are dynamic: when the control system initializes the hardware interface, the composition of the robot does not change. However, controllers are dynamically loaded or unloaded depending upon the activity performed by the robot. When a controller is loaded, it requesta hardware resources from the controller manager. Hardware resources are generally described in terms of control interfaces, *e.g.*, a controller that issues torque commands could ask for an effort command interface which is composed of state information (position, velocity, and effort), as well as a command (effort command).

4.2 Bandwidth Manager

It was known *a priori* that the DRC Trials would provide interrupted and delayed network bandwidths, similar to what would exist in true field-deployment scenarios. Such situations are characterized by limited bandwidth, high latency and intermittent packet loss which can render standard approaches useless. Several approaches are taken to maintain functionality in this environment. Data streams are classified by their frequency and function into *reliable* and *unreliable* types. Reliable streams are comprised of small, non-periodic commands such as remote procedure calls that require acknowledgment of receipt. Unreliable streams are comprised of small, high-rate or large, low-rate periodic messages where a dropped message is acceptable because the message interval is smaller than the time required to retransmit a dropped message. Examples include image streams, joint positions, and error status messages.

The bandwidth manager was designed and implemented to throttle and multiplex the cloud of messages transmitted between robot and operator. Reliable streams are sent over a ZeroMQ (ZMQ) (iMatix Corporation, 2007) link and unreliable streams are sent over an RTP (UDP-based) link. An estimate of the channel bandwidth and of the current bandwidth is used to determine when to send each packet. Each instance of the bandwidth manager maintains an estimate of the local$\rightarrow$remote bandwidth by sending periodic time-stamps and deltas to the remote instance. If the delta in latency is seen to be larger than a designated threshold, the bandwidth estimate is reduced. Also, if a time-stamp packet is not received within a set period from the previous time-stamp packet, the bandwidth estimate is reduced. It is important to note that close clock synchronization is required for accurate bandwidth sensing using this approach. In this implementation, a Network

Time Protocol (NTP) daemon is run on the robot, to which the operator computer periodically synchronizes its clock. Towards the goal of minimizing bandwidth usage, all messages are sent through in-line LZO compression. Depending on the size and amount of entropy in the message, the resulting message may actually exceed the original length. To avoid this, any message that could not be compressed is sent raw instead.

4.3 Data Compression Pipeline

Further compression is performed on Valkyrie's sensory data before arriving at the bandwidth manager. Indeed, even after significant degradation this data is still useful to an operator. As a result, Valkyrie's perception pipeline enforces a constant bit-rate—though optionally lossy—encoding algorithm that is an obvious choice given the bandwidth limitations of the DRC Trials. For 2D data streams, the SPIHT encoding scheme was selected for its constant bit-rate and hierarchal information selection (Said and Pearlman, 1996). It was incorporated into the Valkyrie's perception pipeline using the QccPack library (Fowler, 2000). The SPIHT encoding algorithm sends the most important information (i.e. the data with the highest wavelet transform coefficient) first and then proceeds to send finer resolution details as allowed by the bit budget. To provide a unified compression framework the 2D SPIHT compression algorithm was extended to 3D in order to leverage its advantages for point cloud information.

5 Supervisory Control

The complexity of the mechatronic structure of Valkyrie requires that appropriate "top-level" software tools be used to maximize the robot's functional potential. While the sheer number of high-dimensional degrees of freedom and sensory channels provided by the system make direct teleoperation intractable, the state of the art of humanoid autonomy is not yet sufficient to allow anything close to a "hands-off" approach to run-time operation. Instead, a balance of *shared autonomy* must be sought to effectively off-load as much burden from the operator as possible, while still allowing that operator to take control when necessary to either set task goals or course correct. In the remainder of this document, an overview of the Valkyrie approach to shared autonomy is presented in which both the configuration variables and sensory streams are reduced into a manageable set of signals that provide both a sufficient amount of task structure along with appropriate means of adjustment to fit the run-time context of the robot's operating environment.

Figure 8 diagrams the dimensional reduction the Valkyrie *supervisory control hierarchy* imposes on the system. On top of the robot and its environment, there are two pathways of reduction. The first imposes structure on the command and control variables—the *configuration space*—of the robot. On top of this space, the whole body control framework, as described in Section 5.2, resides in which the command space of the robot is divided up into discrete sets of tasks and constraints that effectively embody the piecewise *intentions* of the robot. This reduction filters the large number of continuous data streams that would be required with direct joint level control of the robot into a smaller (though still highly expressive) set of task goals. As will be described, these goals can be combined in hierarchical, prioritized fashion to achieve a large amount of human understandable goal-directed behavior.

A second pathway of reduction is necessary to filter the large number of data streams published

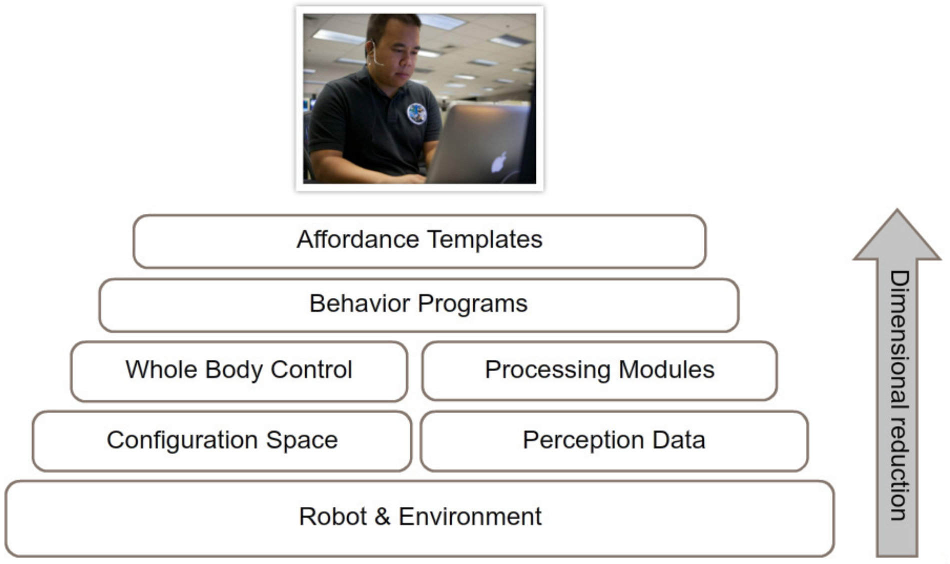


Figure 8: Supervisory Control Hierarchy

by Valkyrie's sensing devices (Section 2.5) into perceptual cues that can be used as either feedback signals to the control tasks or to provide appropriate situational awareness to the operator. It is important to note that this dimensional reduction of sensory streams is necessary not only for creating a more manageable set of data signals, but also because the limited bandwidth and transmission constraints of Valkyrie's operating environment require it from inception (as described in Section 4.3). The pathway of perceptual reduction is achieved by various *processing modules* that compress sensory data in into human parse-able shapes (planes, ellipsoids, etc.) that can effectively be assigned by the operator various control affordances (grasp-ability, traverse-ability, etc.).

Tying these two pathways together, the robot *behavior program* layer ties together the tasks and constraints of the control pathway with the processing modules of the perception pathway. The behavioral programming syntax is then introduced, along with a new IDE called Robot Task Commander (RTC) used to create programs using this syntax. RTC provides an easy-to-use development environment for creating new state machines out of sequences of whole body control tasks. While still an engineering tool for development, RTC allows operators to create a significant amount of task structure while still providing the "knobs" to turn if control signals need to be tweaked or modified at run-time. The final level of the hierarchy introduces a novel programming tool called an *affordance template* that allows the operator to adjust behavior program references in an immersive 3D environment that brings together robot state and sensory data that enables fast modification, recovery, and responsiveness.

5.1 Sensor Processing Modules

Relating the environment and an affordance template poses a challenge, due to the complexity and volume of the raw data, as well as the general nature of the template. Thus Sensor Processing Modules (SPM) were developed to act as a bridge between raw data and affordance templates. SPMs create this bridge by semantically labeling regions of perception data with control oriented labels. This data labeling provides a robust relationship between affordance templates and the environment because primitive shapes are easy to detect, and graphs of them can be used to describe affordance templates (Schnabel et al., 2008). In addition, operators have a reduced mental load due to the increased saliency of control objectives in the environment. For example, the creation of a map from depth data, reduces the information from each scan into an easily consumable object that the operator can act upon.

Primitive shape decomposition performs the semantic labeling that is required by the SPM architecture. The decomposition assumes that the world consists of two types of shapes: planes and ellipsoids. Planes are estimated by performing Principal Component Analysis (PCA) on the points that are found in a certain volume. The axes returned from the PCA are then used to filter the volumes by the following heuristics:

$$\lambda_2 \div \lambda_1 < 0.01 \tag{3}$$

$$0.9 < \lambda_0 \div \lambda_1 < 1.1 \tag{4}$$

Volumes that have been determined to be non-planar are then fit with an ellipsoid using convex optimization (Asokan and Platt, 2014). These shapes were chosen because of the general applicability to the tasks which Valkyrie must perform. For example, ellipsoids properly encode enveloping grasps, as was demonstrated by (ten Pas and Platt, 2014), and planes capture the height and normal of the ground upon which the robot will walk. These primitive descriptions of the world allow for a level of abstraction to be applied via the affordance templates, as described in Section 5.5.

As seen in Figure 9, primitive shape decomposition creates a simplified environment that allows for generalized higher level functions to be applied or simplified shared control. In this image, an image of a double door is shown. Points on the left are shown untouched. Points on the right are converted to planes (shown in red) and ellipsoids (shown in green). While one ellipsoid is (reasonably) found at the door handle, two other spurious ellipsoids are found at the top and bottom of the wall. Although not useful behaviorally, an operator could easily prune these spurious matches with minimal inconvenience.

5.2 Whole Body Control

The Whole Body Control (WBC) framework allows robot programmers to build sophisticated applications out of closed-loop control components (Sentis and Khatib, 2005b; Sentis et al., 2013). In a sense, these components capture the discrete behavioral *intentions* of the system, either as a *task* that tracks a desired reference signal (e.g., a Cartesian end-effector trajectory, a grasp force on an object, etc.), or a *constraint* that ensures a condition is maintained (e.g., contact between the robot and its environment, relative configurations of coupled joints, etc.). Collectively these components are called *WBC primitives*, and can be combined into composite structures called *compound tasks* and *constraint sets*, respectively, that achieve a desired, prioritized ordering. On Valkyrie, a whole-body torque controller is used that computes a dynamically consistent command signal based on the goals of WBC primitives, the current robot configuration, and the mechanical description of that robot (called the *robot model*) (Sentis and Khatib, 2005a). In the remainder of this section, the robot model and WBC primitives are discussed in more detail.

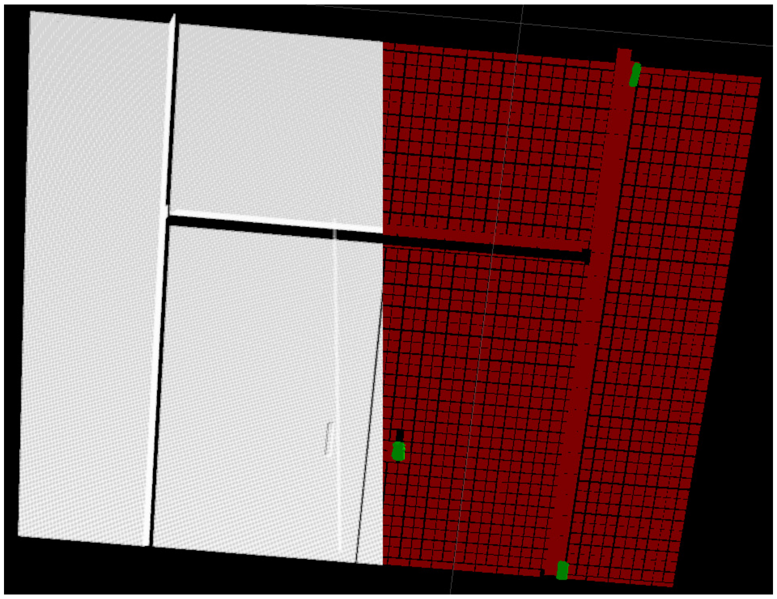


Figure 9: On the left is the raw point cloud data, on the right is the plane representation as well as an ellipsoid fit to the hinges and door handles

5.2.1 Robot Model

The robot model contains the dynamic and kinematic properties of the robot (i.e. a description of the robot's links and the joints that connect them), and is the basis of all WBC primitives. For Valkyrie, a 6-DOF virtual joint is also included in order to capture the robot's position and orientation with respect to the world, as determined from IMU odometry[1]. During execution, the robot model is updated using the current state of the robot (i.e. joint positions, $\boldsymbol{q}$, and velocities, $\dot{\boldsymbol{q}}$). For convenience, the robot's state is defined as

$$S_q := \{\boldsymbol{q}, \dot{\boldsymbol{q}}\}. \tag{5}$$

Note that $\boldsymbol{q}, \dot{\boldsymbol{q}} \in \mathbb{R}^n$ where n is the number of real and virtual DOFs. From this information, various useful dynamic and velocity-dependent quantities such as the joint space inertia matrix, $\boldsymbol{A}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$, the gravity vector, $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$, or Jacobians, $\boldsymbol{J}_{\boldsymbol{\mathcal{F}}}(\boldsymbol{q})$, for an arbitrary frames $\boldsymbol{\mathcal{F}}(\boldsymbol{q})$ such that

$$\dot{\boldsymbol{\mathcal{F}}} = \boldsymbol{J}_{\boldsymbol{\mathcal{F}}} \cdot \dot{\boldsymbol{q}} \in \mathbb{R}^6 \tag{6}$$

can be defined. For simplicity, $\boldsymbol{J}_{\boldsymbol{\mathcal{F}}}$ from Equation 6 will be used to denote both the analytic and geometric Jacobians, with the distinction being that the angular part of the frame velocity is represented in terms of the rate of change of the Euler parameters in the first case, and the angular velocity in the second (Sentis et al., 2010).

[1]Note that for fixed-base robots, the same virtual joint abstraction is used by simply fixing a 6-DoF constraint to the base body.

5.2.2 WBC Primitives

WBC primitives represent the fundamental units of WBC computation from which tasks and constraints are derived. Let us define a WBC primitive as

$$\boldsymbol{p}(\mathcal{S}_q, \boldsymbol{p}_d(\mathcal{S}_q)), \tag{7}$$

where $\boldsymbol{p}_d(\mathcal{S}_q)$ is the desired state of the primitive, possibly dependent on robot's state. As such, $\boldsymbol{p}$ can be generally thought of as an error function. The state of a WBC primitive is calculated from the robot model, the current state of the robot, the current desired state of the WBC primitive, and the Jacobian $\boldsymbol{J_p}(\boldsymbol{q})$, and is defined as

$$S_p = \{\boldsymbol{p}, \dot{\boldsymbol{p}}\}\,. \tag{8}$$

In the following discussion, the notation $\boldsymbol{p}_c$ and $\boldsymbol{p}_t$ will be used to denote the states of constraints and tasks respectively.

Constraints: A constraint encapsulates a mathematical description of a mechanical interaction which is not directly described in the robot model when $\boldsymbol{p}_c = \boldsymbol{0}$. These include contact interactions with the environment (e.g., keeping a foot on the ground, or a hand on a wall), internal mechanical transmissions (e.g., coupled degrees of freedom), and joint limits. Constraints represent the immutable goals of the system under which all other tasks must function. Table 3 provides example constraints that have been implemented in WBC and tested on Valkyrie.

Tasks: WBC tasks are primitive closed-loop control objectives that produce a desired output when $\boldsymbol{p}_t = \boldsymbol{0}$. Tasks using feedback signals based on the robot's state can keep the robot well conditioned; for example, away from joint limits. Tasks using signals derived from the environment can allow the robot to reach out to desired goal locations and interact with objects and tools. Table 4 provides a number of WBC tasks implemented and tested on Valkyrie. These include tasks that control the joint configuration of the robot (JPos), the Cartesian position of a point on the robot (CartPos) relative to a specified frame, the orientation of a link on the robot specified as a Quaternion (QuaternionOrientation), and the desired center-of-mass (COM) of a robot.

5.2.3 Composite Structures

WBC primitives can be combined into composite structures that aim to achieve multiple objectives at a given time. In particular, tasks can be combined into compound tasks that achieve more complex behavior in a prioritized fashion. While certain tasks may reside at the same priority level, there is no guarantee that the steady state error for these tasks will be achieved as progress towards the goal of one may interfere the progress towards another. If a strict ordering is necessary, tasks can be combined using null space projection such that they only execute subject to the goals of other tasks (Nakamura, 1991; Khatib, 1987). Constraints can be combined into constraint sets. However, because constraints represent immutable objections, there is no sense of prioritization as exists in composite tasks. Tasks and composite tasks can also run subject to constraints or constraint sets without interfering with the constraint goals.

5.2.4 Valkyrie Implementation

The WBC framework was successfully implemented for the Valkyrie project, both in simulation and on the real robot. In simulation, WBC was used to control a full representation of the robot in conjunction with planners for walking using inverted pendulum dynamics, similar to that given in (Zhao et al., 2013). On the real robot, the WBC framework was used only to control the upper torso (the robot's arms and head), while the walking functionality was provided by the IK algorithm provided in Section 5.3. This "dual-robot" strategy was implemented for simplicity, as a first pass at integrating manipulation and mobility on the complex Valkyrie system. Ongoing work is investigating a full-robot WBC implementation on the hardware system.

The WBC library was written in C++ and was initially implemented in a single-threaded manner where the model and task Jacobians were updated every cycle of the servo loop in series with the computation of the joint commands via a series of dynamically-consistent null space projections. Later, to achieve higher servo frequency, the process of updating the model and task Jacobians was offloaded into separate child threads. This is possible because the joint states change slowly relative to the servo frequency. Note that in the multi-threaded architecture the latest joint state information is still used in two places: (1) in the error calculations of the JPos task that specifies the robot's posture and is present as the lowest-priority task in every compound task used with Valkyrie, and (2) in the impedance controller that converts the torque commands produced by the WBC algorithm into an impedance command consisting of a desired torque, position, and velocity for each joint. Having a high servo frequency is desirable because it enables higher controller gains and thus increases the achievable levels of tracking accuracy and impedance.

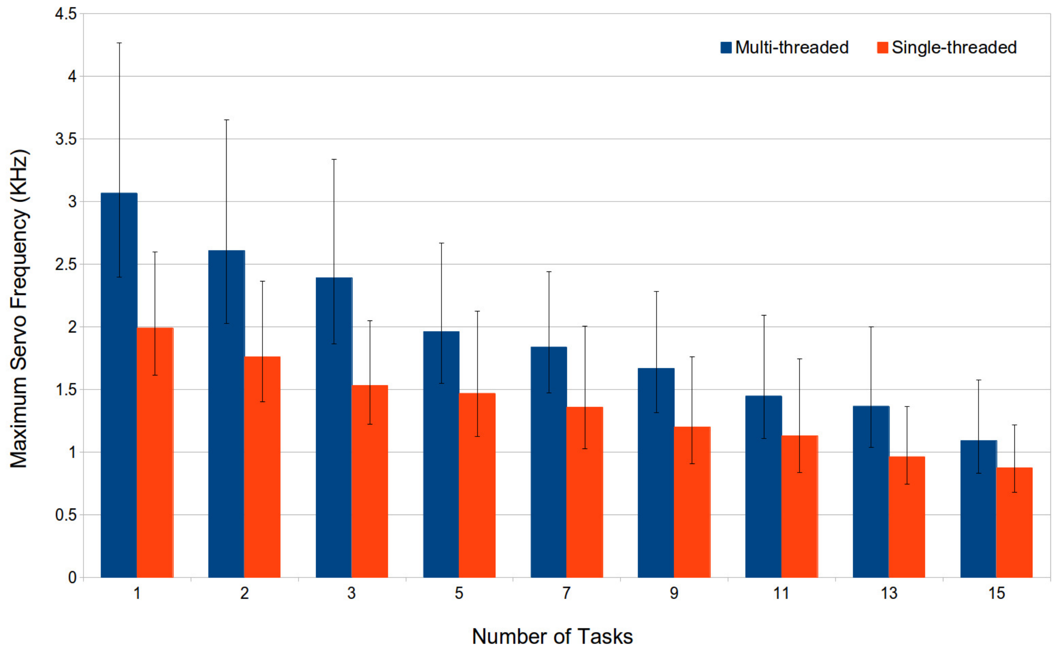


Figure 10: The maximum servo frequency of WBC versus the number of tasks when controlling Valkyrie's 17 DOF upperbody.

The maximum servo frequency depends on the computational complexity of the WBC controller, which in turn depends on complexity of the compound task. To understand the maximum servo frequency that can be achieved by our implementation, we measure the latency of executing one cycle of the servo loop with compound tasks containing between one and fifteen tasks. This is because the largest compound task we used for the DRC Trials contained fifteen tasks. We then invert these latency measurements to obtain the theoretical maximum frequency that can be obtained. Servo frequency was not directly measured since the controller operates asynchronously from the robot and servo loop is triggered by a periodic clock of known frequency (1kHz in Valkyrie's case). The purpose of this study is to gain insight into the maximum frequency that could be achieved assuming no other bottlenecks in the system, e.g., latencies from communication, sensing, and actuation. For each compound task, we measured the execution latency of the servo loop ten thousand times. All measurements were done on a laptop containing an Intel Core i7-3520M 2.90GHz CPU and 16GB of DDR3 RAM, running Ubuntu Linux 12.04.4 and Linux kernel 3.13.0 (a non-realtime OS was used in this study because the same OS was used on Valkyrie during the DRC Trials). The WBC controller was configured to use Valkyrie's 17 DOF upperbody. The results are shown in Figure 10. Each vertical bar denotes the average and the error bars denote the standard deviation of the dataset. As indicated by the figure, the multi-threaded control architecture increases the maximum achievable servo frequency over the single-threaded implementation by $25-54\%$ depending on the size of the compound task. Servo frequencies in the range of 1kHz to 3kHz are achievable by our WBC implementation. This is sufficiently high to achieve the necessary performance for the DRC tasks.

5.3 Inverse Kinematics (IK) Locomotion Strategy

During the DRC Trials, only Valkyrie's upper body ran under impedance mode, receiving its commands from WBC. Valkyrie's lower body, however, ran in position mode, receiving desired trajectories from an inverse kinematics (IK) based walking algorithm. This hybrid approach was taken due to the implementation challenges involved with model-based locomotion on a robot of Valkyrie's complexity.

When designing controllers for stable biped locomotion, a highly accurate dynamic model of the robot is not always available. Although the structure of the dynamic equations of motion are well known and model parameters such as link lengths and total robot mass may be known very accurately, other parameters will likely not be known as accurately. Values for link inertia tensors, encoder offsets, coefficients of friction, elastic effects of cabling, power supply dynamics, force sensor calibration, etc. are typically much more difficult to measure in practice. As mentioned in (Mistry et al., 2008), model-based techniques suffer when there is large model error. In situations where model error is known or suspected to be a limiting factor in achieving locomotion, a position-based controller may be preferred over a similar torque-based model dependent controller. Inverse kinematics is a popular technique that can be employed to produce model-independent position based control. This section focuses on the non-redundant Newton-Raphson inverse kinematics technique, which is largely dismissed in the literature (Tevatia and Schaal, 2000; Whitney, 1972; Mistry et al., 2008); however, due to the algorithms simplicity, it can lead to quick prototyping of simple behaviors as a robotic system is being developed.

5.3.1 IK Algorithm

Given a set of actual and desired task values, $\boldsymbol{y}_a(\boldsymbol{q})$ and $\boldsymbol{y}_d(t)$, inverse kinematics tries to find a value, $\boldsymbol{q}_d$, such that $\boldsymbol{y}_a(\boldsymbol{q}_d) = \boldsymbol{y}_d(t)$ within a specified tolerance, ϵ. Since PD-control was employed,

Algorithm 1 Inverse Kinematics

```
Inputs: $\boldsymbol{q}, t, \boldsymbol{y}_a(\boldsymbol{q}), \boldsymbol{y}_d(t), \boldsymbol{q}_d^{seed}, \epsilon$
Outputs: $\boldsymbol{q}_d$, $\dot{\boldsymbol{q}}_d$
  $\boldsymbol{q}_d \leftarrow \boldsymbol{q}_d^{seed}$
  $\boldsymbol{y}_{err} \leftarrow \boldsymbol{y}_a(\boldsymbol{q}_d^{seed}) - \boldsymbol{y}_d(t)$
  while $\max(|\boldsymbol{y}_{err}|) > \epsilon$ do
      $\boldsymbol{J} \leftarrow \frac{\partial \boldsymbol{y}_{err}}{\partial \boldsymbol{q}}(\boldsymbol{q}_d)$
      $\boldsymbol{q}_d \leftarrow \boldsymbol{q}_d - \boldsymbol{J}^{-1}\boldsymbol{y}_{err}$
      $\boldsymbol{y}_{err} \leftarrow \boldsymbol{y}_a(\boldsymbol{q}_d) - \boldsymbol{y}_d(t)$
  end while
  $\boldsymbol{J}_a \leftarrow \frac{\partial \boldsymbol{y}_a}{\partial \boldsymbol{q}}(\boldsymbol{q}_d)$
  $\dot{\boldsymbol{q}}_d \leftarrow \boldsymbol{J}_a^{-1}\dot{\boldsymbol{y}}_d(t)$
```

the derivative term, $\dot{\boldsymbol{q}}_d$, was also included as an output of the algorithm. The iterative Newton-Raphson method is shown as a modified version of the IK algorithm from (Meredith and Maddock, 2004) in **Algorithm 1**, and is used to obtain the vector of desired joint positions, $\boldsymbol{q}_d$, at a given point in time.

This implementation can be stated with the functional relationship $(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d) = \text{IK}(\boldsymbol{y}_a(\cdot), \boldsymbol{y}_d, \boldsymbol{q}_d^{seed})$. Assuming continuity in the output trajectories, using the previously computed inverse kinematics value as the seed value, $\boldsymbol{q}_d^{seed}$, can produce quicker convergence of the Newton-Raphson algorithm. Note that $\boldsymbol{J} = \frac{\partial \boldsymbol{y}_{err}}{\partial \boldsymbol{q}} = \frac{\partial \boldsymbol{y}_a}{\partial \boldsymbol{q}} = \boldsymbol{J}_a$ since $\boldsymbol{y}_d$ is a function of time and not state. Also note that the coordinates used will be the full, extended coordinates, $\boldsymbol{q}$, which is comprised of the virtual or floating-base coordinates, $\boldsymbol{q}_b \in \mathbb{R}^6$, and the physical robot generalized coordinates, $\boldsymbol{q}_r \in \mathbb{R}^n$, where n is the number of generalized coordinates for the robot.

5.3.2 Output Selection

The selection of the constraints or outputs, $\boldsymbol{y}_a$, is an important step in the implementation of different motion behaviors. The constraints for the walking behavior were inspired by the outputs used in dynamic (hybrid zero dynamics-based) locomotion, with examples of such constraints given in (Grizzle et al., 2010; Morris et al., 2013; Lack et al., 2014; Ames et al., 2012). In particular, outputs were chosen to regulate the full-body behavior of the robot, including the center of mass, foot height, lateral foot spacing, etc. For stepping over a cinder block, additional constraints were placed on the torso and waist orientation. For the floating-base model, where the torso is the base link, constraints were supplied to fix the position and orientation of the feet, which then assumes that the feet are flat upon the ground. The specific outputs used in these behaviors are simply sets of linear combinations of nonlinear Cartesian positions in the world frame and linear combinations of the generalized coordinates. The desired outputs were time-based desired trajectories with a set of parameters, $\boldsymbol{\alpha}$, resulting in the form $\boldsymbol{y}_d(t, \boldsymbol{\alpha})$. In this case, they were defined as a minimum-jerk profile which goes from an initial position to a final position, beginning and ending with zero velocity and zero acceleration.

5.3.3 Planning

Several different trajectories or motion primitives must be pieced together to realize behaviors as complex as walking or walking over cinder blocks. Given the simplified inverse kinematics technique, basic steps need to be taken to ensure that the behaviors produced are continuous and, if needed, periodic. In order to enforce continuity, a planner needs to operate between two different sets of constraints. When the planner switches between the current set of constraints, $\boldsymbol{y}_a^c(t)$ with the desired trajectories specified as $\boldsymbol{y}_d^c(t, \boldsymbol{\alpha}^c)$, to the next set of constraints, $\boldsymbol{y}_a^n(t)$ and $\boldsymbol{y}_d^n(t, \boldsymbol{\alpha}^n)$, the planner must provide a valid set of parameters, $\boldsymbol{\alpha}^n$, for the new set of constraints, such that $\mathrm{IK}(\boldsymbol{y}_a^c(\cdot), \boldsymbol{y}_d^c(t_f^p, \boldsymbol{\alpha}^c), \boldsymbol{q}_d^{seed}) = \mathrm{IK}(\boldsymbol{y}_a^n(\cdot), \boldsymbol{y}_d^n(t_0^n, \boldsymbol{\alpha}^n), \boldsymbol{q}_d^{seed})$, where t_f^p is the end-time of the current set of constraints and t_0^n is the initial time for the next set of constraints, and $\boldsymbol{q}_d^{seed}$ in this case is the previously solved-for inverse kinematics value. However, due to the nature of the deadbeat update, drift may occur in the inverse kinematics solutions and may cause aperiodicity, namely in the floating-base coordinates. To avoid this drift, certain states must be specified where all of the desired values are captured and restored in subsequent visitations. This will enforce periodicity due to the full-rank nature of the inverse kinematics problem.

5.3.4 Valkyrie Implementations

The non-redundant Newton-Raphson algorithm and the planner concepts introduced above were implemented and used to experimentally achieve prototype walking and cinder block climbing behaviors. Statically stable walking was implemented with the aforementioned set of outputs, with Figure 11 showing tiles of the behavior on the physical hardware. The experimental tracking plots in Figure 12 show very low tracking error, indicating excellent performance in the hardware. A video of this walking behavior experiment is publicly available online[2]. The cinder block behavior, where Valkyrie takes a total of four steps up and over a row of cinder blocks, was implemented in the same fashion as the walking behavior. The resulting behavior is shown in Figure 13, and the experimental tracking plots are shown in Figure 14. A video of this cinder block behavior experiment is publicly available online[3].

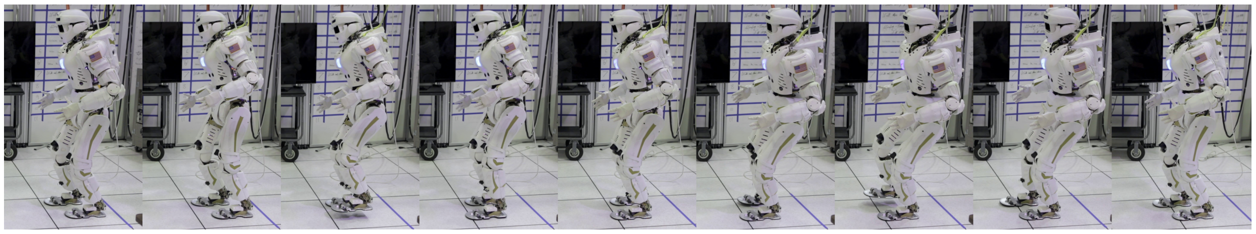


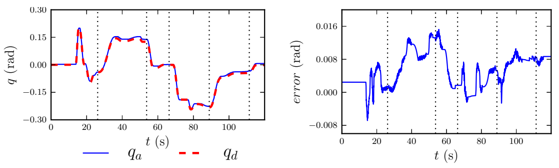
Figure 11: Tiles of walking behavior in action.

5.4 Robot Task Commander

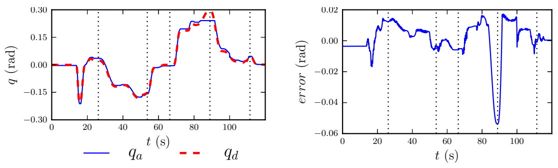
Robot Task Commander (RTC) is a robot programming environment that allows task developers to build applications out of controllers and sensory processing modules to perform complex tasks in

[2]`http://youtu.be/hxvYLO713Cc`

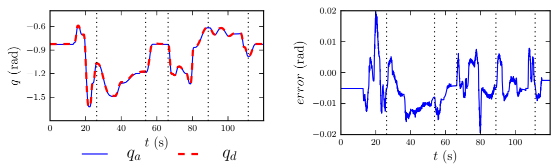
[3]`http://youtu.be/qubDKVCut4o`



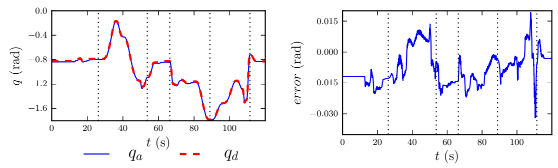
(a) Left Hip Adductor



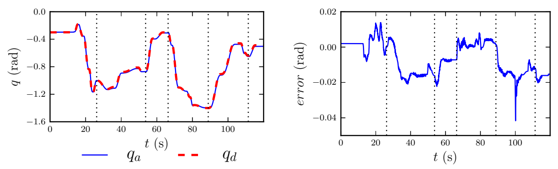
(b) Right Hip Adductor



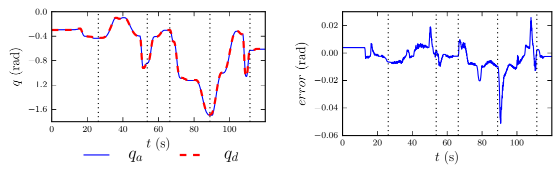
(c) Left Knee Extensor



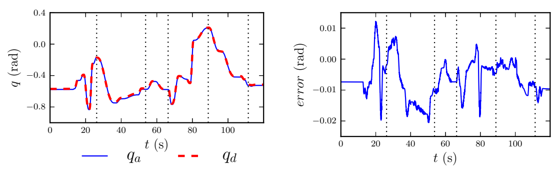
(d) Right Knee Extensor



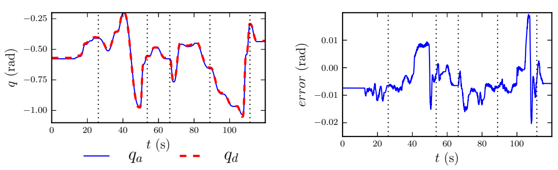
(e) Left Hip Extensor



(f) Right Hip Extensor



(g) Left Ankle Extensor



(h) Right Ankle Extensor

Figure 12: Plots of the actual $\boldsymbol{q}_a$ versus desired $\boldsymbol{q}_d$ joint angle trajectories from the walking experiment with error metrics. Each of the vertical lines represents a major transition in the state machine. Note that the ankle angles are computed from a transmission mapping coupled linear actuators to pitch and roll angles.

real-world environments (Hart et al., 2014b). RTC provides an integrated development environment (IDE) that is suitable for both development and deployment of applications that can respond to different, possibly unforeseen, situations either autonomously or by enabling operators to modify these programs quickly in the field. While more fully autonomous solutions to complex robot behavior still tend to be relegated to the laboratory, run-time application modification usually requires a certain amount of expert knowledge (i.e. re-programming by a robotics expert or system designer). This can be an expensive and potentially time-consuming practice. However, with appropriate tools and levels of abstraction, robot applications can be adapted and modified quickly. To these ends, RTC has been designed for use by both experts and non-experts to quickly create, re-use, and adapt application software for complex systems such as humanoid robots or a multi-robot manufacturing plant.

5.4.1 RTC Design Principles

The following design principles were followed in the development of the RTC framework:

1. Robot applications programs must adhere to an application syntax that also allows for hierarchical composition.

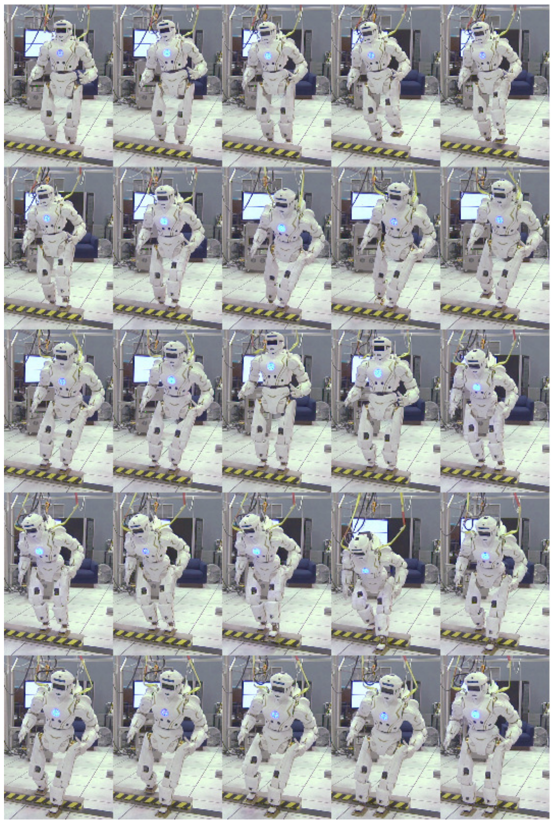


Figure 13: Tiles of cinder block behavior in action.

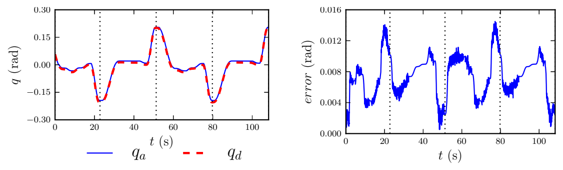
2. Applications must control the flow of both distributed computation as well as the designation of a robot's control mode.

3. Units of computation must be generalizable and, therefore, must be written in terms of abstract interfaces (i.e. *types* of data), not specific robot resources.

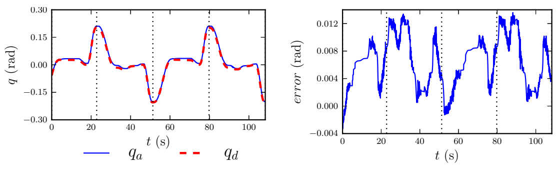
4. All functionality must be stored in a library accessible for re-use to minimize code recreation.

5. Appropriate interfaces must exist for both experts and non-experts.

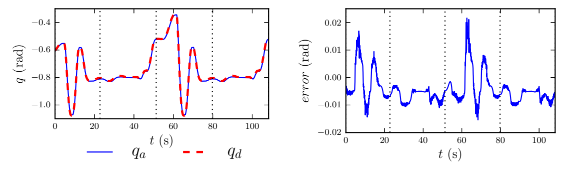
Together, these principles emphasize a paradigm of application flexibility and re-usability. A formal syntax will encourage principled construction that minimize brittle code that leads to fast deprecation and "bit-rot." Computation that is written generally in terms of data types (joint positions, Cartesian positions, wrenches, etc.) instead of specific hardware resources (Valkyrie's left arm or right foot), allows behavior to be re-parameterized in different contexts. For example, an RTC program for pick-and-place can be re-parameterized to use Valkyrie's left arm or right arm at the operator's run-time request. By ensuring that all programs are stored in a library and composable hierarchically, functionality can be re-used easily. A pick-and-place program, for example, can be used in a more complex stacking program. Finally, while expert interfaces provide an advanced developer to code complex algorithms (RTC provides a Python language interface for building computation components called *process nodes*), more stream-lined interfaces, like a visual programming



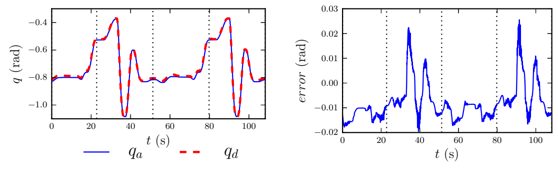
(a) Left Hip Adductor



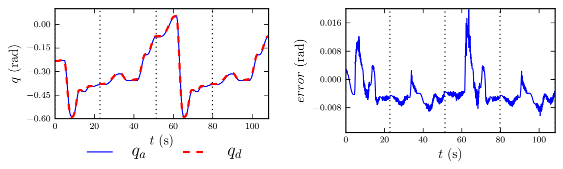
(b) Right Hip Adductor



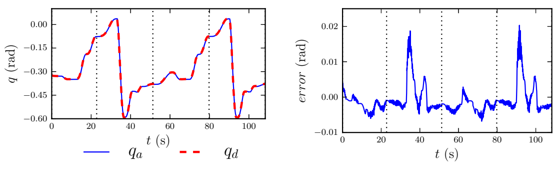
(c) Left Knee Extensor



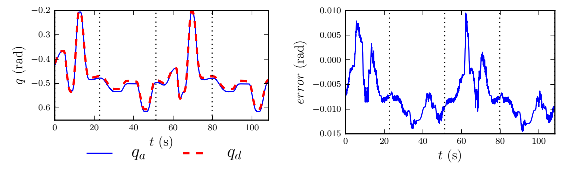
(d) Right Knee Extensor



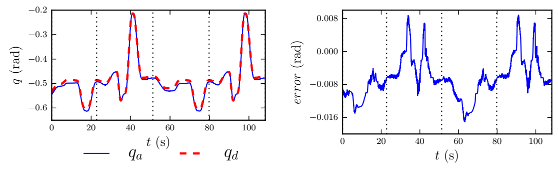
(e) Left Hip Extensor



(f) Right Hip Extensor



(g) Left Ankle Extensor



(h) Right Ankle Extensor

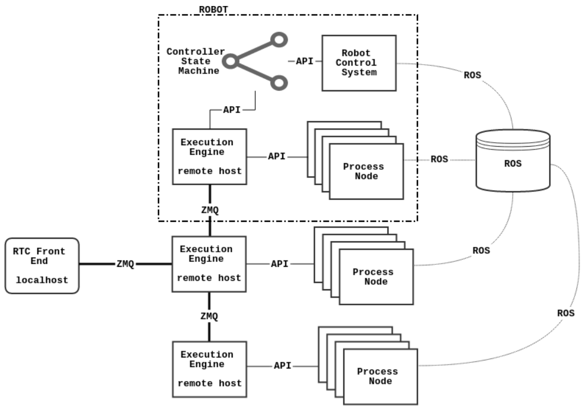
Figure 14: Plots of the actual q_a versus desired q_d joint angle trajectories from the cinder block experiment with error metrics, formatted the same as Figure 12.

language (VPL) are more suitable for non-expert users or to facilitate fast operator modification at run-time, when a code-level interface is too cumbersome.

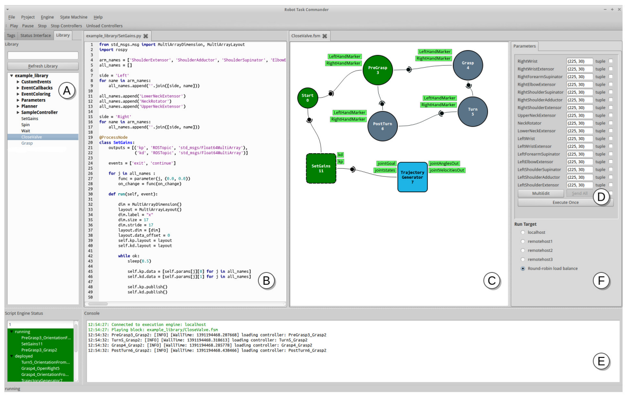
5.4.2 RTC Implementation

The RTC system architecture (Figure 15(a)) consists of a single graphical *front end* and at least one *execution engine*. The front end, shown in Figure 15(b), represents a simple and unified point of access for an operator to develop, deploy, or modify applications that control a potentially large number of remote systems. The execution engines handle the execution of each application component—*process nodes* (such as sensory processing modules) that perform auxiliary computation, and *control state machines* that sequence and parameterize composite WBC tasks and constraints—as well as communication between itself, other engines, and the front end.

Decomposing the deployment architecture into multiple execution engines and a single user interface presents a key advantage in limited bandwidth scenarios such as controlling Valkyrie. In RTC, all development is done locally on the user's machine, but during deployment, only basic program flow feedback and control is required; discrete information that can typically be transmitted even with severe bandwidth limitations. Additionally, user feedback from the remote engine(s) to the graphical front end can be manually configured during runtime, allowing the user to dynamically prioritize which feedback data, if any, he or she wants during operations. To further keep the bandwidth usage low, all data is automatically compressed using LZMA (Merhav et al., 1989) and then serialized using the Python cPickle library before being sent across a socket using ZMQ.



(a)



```python
from std_msgs.msg import MultiArrayDimension, MultiArrayLayout
import rospy

arm_names = ['ShoulderExtensor', 'ShoulderAdductor', 'ShoulderSupinator', 'ElbowE
all_names = []

side = 'Left'
for name in arm_names:
    all_names.append(''.join([side, name]))

all_names.append('LowerNeckExtensor')
all_names.append('NeckRotator')
all_names.append('UpperNeckExtensor')

side = 'Right'
for name in arm_names:
    all_names.append(''.join([side, name]))

@ProcessNode
class SetGains:
    outputs = [('kp', 'ROSTopic', 'std_msgs/Float64MultiArray'),
               ('kd', 'ROSTopic', 'std_msgs/Float64MultiArray')]

    events = ['exit', 'continue']

    for j in all_names :
        func = parameter(j, (0.0, 0.0))
        on_change = func(on_change)

    def run(self, event):

        dim = MultiArrayDimension()
        layout = MultiArrayLayout()
        dim.label = "x"
        dim.size = 17
        dim.stride = 17
        layout.dim = [dim]
        layout.data_offset = 0
        self.kp.layout = layout
        self.kd.layout = layout

        while ok:
            sleep(0.5)

            self.kp.data = [self.params[j][0] for j in all_names]
            self.kd.data = [self.params[j][1] for j in all_names]

            self.kp.publish()
            self.kd.publish()
```

```
12:54:27: Connected to execution engine: localhost
12:54:27: Playing block: example_library/CloseValve.fsm
12:54:32: PreGrasp3_Grasp2: [INFO] [WallTime: 1391194468.207668] loading controller: PreGrasp3_Grasp2
12:54:32: Turn5_Grasp2: [INFO] [WallTime: 1391194468.318613] loading controller: Turn5_Grasp2
12:54:32: Grasp4_Grasp2: [INFO] [WallTime: 1391194468.285778] loading controller: Grasp4_Grasp2
12:54:32: PostTurn6_Grasp2: [INFO] [WallTime: 1391194468.438466] loading controller: PostTurn6_Grasp2
```

(b)

Figure 15: (a) shows the RTC Architecture. Multiple execution engines can be deployed on the network to run various processing nodes that communicate using ROS. On the robot, a single execution engine communicates with the control system to handle time-critical operations such as controller switching and activation. (b) shows the RTC Front End. This graphical user interface provides access to the RTC application library (**A**), a text editor for writing new Python process nodes (**B**), a canvas to build hierarchical nodes and applications using a VPL (**C**), a parameter editor for the selected state or node (**D**), a command output window streaming relevant run-time information (**E**), and a deployment manager (**F**) that allows the operator to select the execution engine a selected node will execute on.

5.5 Affordance Templates

Although a significant amount of task structure can be programmed in RTC applications (transitions between WBC control modes, sensory processing modules, etc.), it is important to provide a flexible interface for the operator to set or adjust additional task parameters at run-time in order to ensure task success. These parameters may include controller reference goals or trajectories, object locations or poses, navigational way-points, etc. This paradigm follows a methodology of shared control, which lets the operator and robot control different signals simultaneously, rather than on traded control, which assumes direct teleoperation or full autonomy. This notion of shared control can reduce the operator's workload and is therefore suitable over unreliable and/or unknown networks. Shared control describes a system in which an operator commands higher-level goals as opposed to every joint-level robot command.

The shared control level of the Valkyrie system follows a supervisory control paradigm of *Plan, Teach, Monitor, Intervene, and Learn* (PTMIL) (Sheridan, 1992). During the planning step, the operator understands the robot's environment and what objectives need to be satisfied in the control loop. Specifically, an explicit link between the run-time context and the robot's task parameters is created by allowing the operator to identify, through the robot's sensory systems, the likely *affordances* in the environment and the controller reference signals that can be used to accomplish the corresponding behavior. An affordance describes a place or object in the world that affords an action by a particular agent. If something affords "sitting" it is not that that thing is a chair,

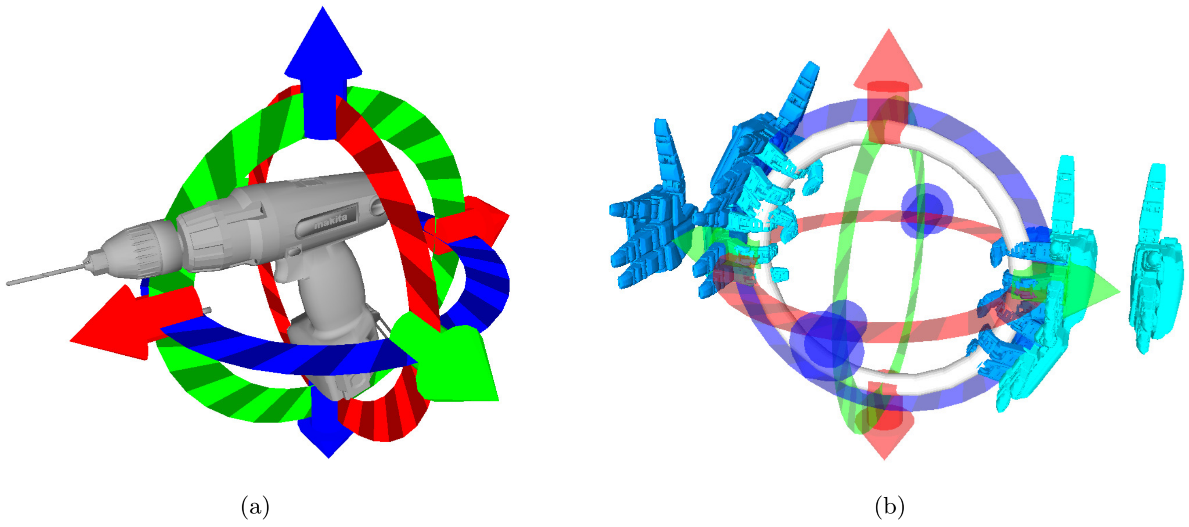


Figure 16: (a) shows a interactive marker with 6-DOFs of control associated with a drill shape. (b) shows a "wheel" affordance template with different Valkyrie hands markers providing controls for various task sub-goals (pre-grasp, grasp, turn-goal, etc.).

but rather that it has a surface that the agent could sit on (without falling off), whether it be a chair, a stool, a rock, or the ground. The concept of affordances was initially introduced in the psychological literature by Gibson (Gibson, 1977) as a means of describing cognitive constructs that assign functional merit to the environment. It is therefore readily applicable for programming robots; embodied agents that must perform actions to achieve some discernible objective (Chemero, 2003; Stoytchev, 2005; Hart and Grupen, 2013).

For Valkyrie, a new programming tool called an *affordance template* was designed specifically for the operator to overlay and adjust 3D representations of RTC program goals in an immersive 3D environment (Hart et al., 2014a). Affordance templates were implemented using the RVIz interactive marker packages (Gossow et al., 2011). Interactive markers consist of visual controls with which a user can interact in an immersive 3D environment that contains robot state feedback and 3D sensory data (laser scans, RGB-D point clouds, occupancy grids). Interaction can occur by dragging the controls with the mouse, or by creating custom menu options that can be accessed by right-clicking the marker. Each individual marker allows up to 6 controllable degrees of freedom (position and orientation and can be associated with custom shapes described by the developer (e.g., a tool, a representation of the robot's end-effector, etc.). Figure 16(a) shows a 6-DOF interactive marker associated with a drill shape model. Multiple interactive marker controls can be associated together into composite structures that allow multiple means of adjustment and are suitable for complex tasks like opening a door and walking through it or using a picking up a fire hose and attaching it to a wye. Figure 16(b) shows a composite template for wheel turning that has multiple hand goal locations (pre-grasp, grasp, turn, release) appropriate for accomplishing the task. By right-clicking on each of the hands, 6-DOF controls can be shown to provide more fine-tuned adjustment.

Figure 5.5 shows the placement of the wheel turning affordance template in the immersive RViz environment. In (a), Valkyrie is shown standing in front of a valve. In (b), the RViz environment showing the state feedback of the robot is seen along with compressed point cloud data (Section 4.3)

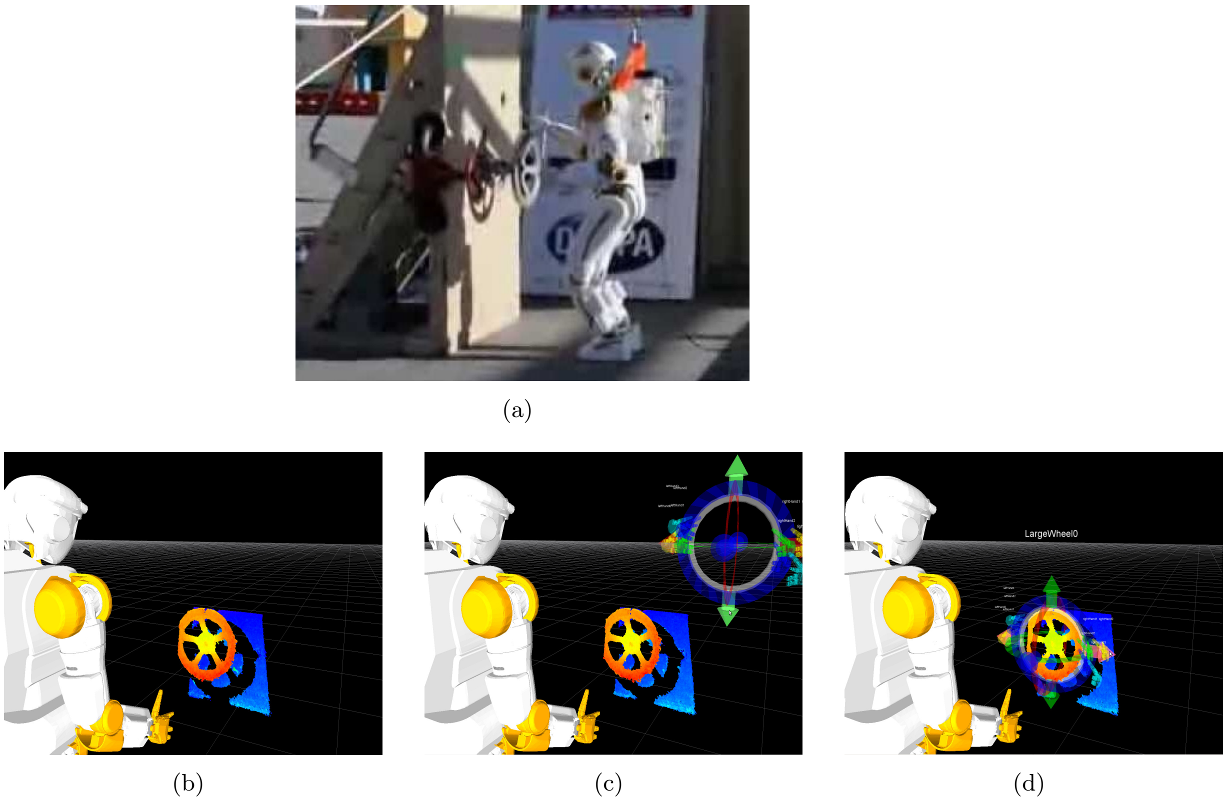


Figure 17: (a) shows Valkyrie standing in front of a valve. (b) shows the RViz view with the robot and a registered scan from the head-mounted point cloud device. (c) and (d) show the wheel template before and after it is overlaid (by the operator) on the sensor data to provide the appropriate goal references for the RTC wheel-turning program.

coming from the head-mounted Ensenso sensor. From this view, an operator can clearly identify the valve in the robot's workspace, and can overlay the wheel affordance template, seen in (c), and adjust the hand pre-grasp, grasp, and grasp goal locations (shown as different color hands, displayed in the wheel reference frame) as necessary. When the operator is satisfied with these goal locations, it can publish the information to an underlying RTC program that sequences the motions accordingly to turn and release the valve (Figure 18).

It should be noted that, over time, affordance templates could monitor task performance and learn from this experience. Template parameters (such as an object's size, it's location in the workspace, etc.) that become reliable predictors of success or failure can be monitored to give the operator feedback either before or during task execution. This feedback could indicate when the operator might need to intervene to make corrections and better ensure task success. This additional functionality, although not currently implemented in the affordance template framework, is fully compatible with the PTMIL paradigm, and is a subject of ongoing investigation.

6 Concluding Remarks

The NASA-JSC team completed the design and development of “clean sheet” robot named Valkyrie in under a 12 month timeline. The resulting fully electric machine is completely self contained including a sophisticated on-board power system and powerful on-board computers. This naturally gives the robot quite a head start toward the eventual and intended use of performing work in remote, austere locations in human engineered environments. The robot design pushed the envelope, advancing technologies on several fronts including high performance series elastic actuators, torque sensing, energy storage, embedded motion control, tactile sensing and electric motors. The software architecture, high level control and operator interface are very powerful tools that were developed first in simulation and then deployed to the robot in an equally aggressive timeline and made significant original contributions to the field.

To mitigate the risk of an accelerated development timeline to develop bipedal locomotion, the original plan called for fabricating two Valkyries that would support the aggressive learning phase of deploying the walking algorithms to the robot. However, due to funding limitations, only one Valkyrie was produced. This caused considerable development strain between the locomotion team and the manipulation and applications team developing the tasks. Because of the modularity of the robot, the robot was divided at the waist to continue development of manipulation tasks on the upper body and development of inverse kinematics based walking on the lower body.

While trial runs proved Valkyrie competent in multiple tasks, and the robot continuously scored seven challenge points during these runs, the robot performed poorly at the DRC Trials in December of 2013, failing to score any points during the actual competition. This can be attributed most directly to four major factors: NASA JSC had very little experience with bipedal locomotion robots, the deployment timeline was too aggressive, not all systems were fully vetted before deployment and the untimely U.S. government shutdown (Office of Management and Budget, 2013).

The next steps for Valkyrie involve improvements to the control software and the operator interface and evaluations of hardware systems, such as the perceptive sensor suite. For For the DRC trials, walking proved non-robust due to the lack of deep sensor integration together with the statically stable locomotion strategy implemented as a result of the time-critical nature of the chal-

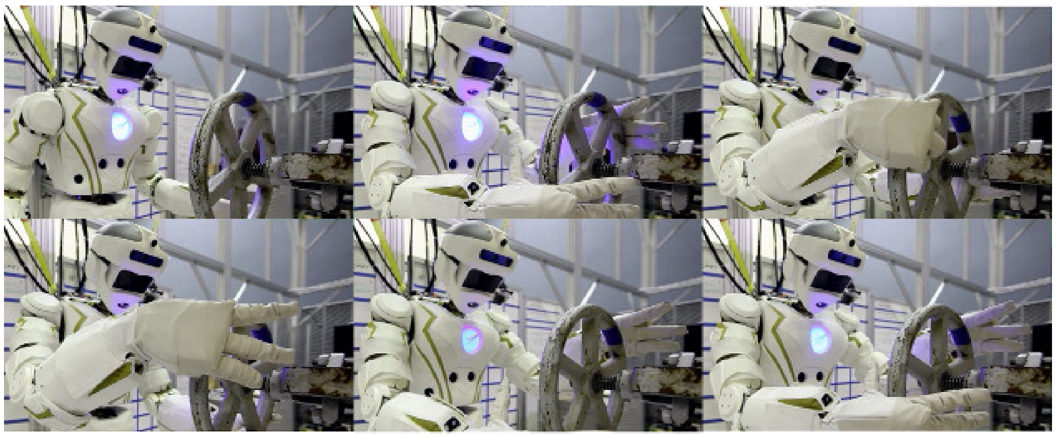


Figure 18: Valkyrie turning a valve using an affordance template.

lenge. Proceeding forward, dynamically stable walking will be implemented, integrating walking and manipulation with WBC will be pursued. In particular, using WBC to implement a more sophisticated, feedback-driven locomotion strategy, such as capture-point walking (Pratt et al., 2006) will be investigated. Manipulation was performed mainly by position controllers; future improvement of manipulation will be attained by the addition of force- or impedance-based controllers that will allow for more compliant, human-safe motions (Platt Jr et al., 2010). The affordance template paradigm will be adapted to provide more flexibility during run-time operations and to offer a larger suite of templates for unknown operations scenarios. Tighter integration between perceived objects in the robot's environment, provided by primitive shape finding SPMs, and affordance templates will be achieved to reduce operator workload by bootstrapping the affordance identification procedure that is currently done fully by the operator.

Long-range plans for Valkyrie and its systems involve operations on other planets as either an astronaut-assistant robot or as a pre-cursor robot for setup. The robots in these scenarios will be expected to perform many of the functions that Valkyrie has and will perform for the DRC. While wheeled mobility may be desirable for operations over planetary surfaces, situations exist where bipedal locomotion is advantageous, such as climbing ladders, traversing human-sized walkways and scaling habitats. Valkyrie is a grand step in the march towards intelligent NASA robotics for space exploration.

7 Acknowledgments

The authors would like to send a special thanks to Dr. Gill Pratt who funded and supported this work and all the staff at DARPA who helped organize and run the DRC event's successful execution. It was an amazing opportunity to be a part of this worldwide endeavor. We would also like to thank our management within NASA's Johnson Space Center and especially Dr. Michael Gazarik, Associate Administrator of the Space Technology Mission Directorate at NASA Headquarters, for supporting this work and additionally funding the development in conjunction with DARPA. We would also like to offer thanks to Laurie Rich, Deputy Director and Special Advisor on Higher Education at the State of Texas, and the State of Texas' Emerging Technology Fund, which funded the Texas universities involvement in NASA's team. We would like to thank Jacobs Engineering for supporting this work with their Innovation Fund and working through the many challenges of delayed government funding. We would also like to thank the many NASA contractors that took part in this challenge with the NASA team. Finally, we would like to thank the University of Texas at Austin and Texas A&M University staff for staying patient as we worked through the many logistical issues that provided NASA access to the talented university faculty and students.

References

Ambrose, R., Aldridge, H., Askew, R., Burridge, R., Bluethmann, W., Diftler, M., Lovchik, C., Magruder, D., and Rehnmark, F. (2000). Robonaut: Nasa's space humanoid. *Intelligent Systems and their Applications, IEEE*, 15(4):57–63.

Ames, A. D., Cousineau, E. A., and Powell, M. J. (2012). Dynamically stable bipedal robotic walking with NAO via human-inspired hybrid zero dynamics. In *Hybrid Systems: Computation and Control*, pages 135–44, Beijing.

Asokan, P. and Platt, R. (2014). Ellipsoid consensus: An approach to localizing handles for grasping in 3d point clouds. In *ICRA*. IEEE.

Bridgwater, L., Ihrke, C., Diftler, M., Abdallah, M., Radford, N., Rogers, J. M., Yayathi, S., Askew, R. S., and Linn, D. (2012). The robonaut 2 hand - designed to do work with tools. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3425–3430.

Chemero, A. (2003). An outline of a theory of affordances. *Ecological Psychology*, 15(3):181–195.

Diftler, M., Culbert, C., Ambrose, R., Platt, R., J., and Bluethmann, W. (2003). Evolution of the nasa/darpa robonaut control system. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 2, pages 2543–2548 vol.2.

Diftler, M., Mehling, J., Abdallah, M., Radford, N., Bridgwater, L., Sanders, A., Askew, R., Linn, D., Yamokoski, J., Permenter, F., Hargrave, B., Platt, R., Savely, R., and Ambrose, R. (2011). Robonaut 2 - the first humanoid robot in space. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Fowler, J. (2000). Qccpack: An open-source software library for quantization, compression, and coding. In *Applications of Digital Image Processing XXIII*, pages 243–250. SPIE.

Gibson, J. J. (1977). The theory of affordances. In *Perceiving, acting and knowing: toward an ecological psychology*, pages 67–82, Hillsdale, NJ. Lawrence Erlbaum Associates Publishers.

Gossow, D., Leeper, A., Hershberger, D., and Ciocarlie, M. T. (2011). Interactive markers: 3-D user interfaces for ros applications [ros topics]. *IEEE Robotics & Automation Magazine*, 18(4):14–15.

Grizzle, J., Chevallereau, C., Ames, A. D., and Sinnet, R. W. (2010). 3d bipedal robotic walking: Models, feedback control, and open problems. In *Proceedings of the IFAC Symposium On Nonlinear Control Systems.*

Guo, R., Nguyen, V., Niu, L., and Bridgwater, L. (2014). Design and analysis of a tendon-driven, under-actuated robotic hand. In *Proceedings of the ASME 2014 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Buffalo, NY. USA. IDETC/CIE.

Harrison, D. A., Ambrose, R., Bluethmann, B., and Junkin, L. (2008). Next generation rover for lunar exploration. In *Aerospace Conference, 2008 IEEE*, pages 1–14.

Hart, S., Dinh, P., and Hambuchen, K. (2014a). Affordance templates for shared robot control. In *Artificial Intelligence and Human-Robot Interaction, AAAI Fall Symposium Series*, Arlington, VA. USA.

Hart, S., Dinh, P., Yamokoski, J., Wightman, B., and Radford, N. (2014b). Robot Task Commander: A framework and IDE for robot application development. In *International Conference on Intelligent Robots and Systems (IROS)*, Chicago, IL. USA. IEEE/RSJ.

Hart, S. and Grupen, R. (2013). Intrinsically motivated affordance discovery and modeling. In Baldassarre, G. and Mirolli, M., editors, *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 279–300. Springer Berlin Heidelberg.

Ihrke, C. A., Mehling, J. S., Parsons, A. H., Griffith, B. K., Radford, N. A., Permenter, F. N., Davis, D. R., Ambrose, R. O., Junkin, L. Q., et al. (2012). Rotary series elastic actuator. US Patent 8,291,788.

iMatix Corporation (2007). ZeroMQ: Distributed Computing Made Simple.

Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Robotics & Automation*, 3(1):43–53.

Lack, J., Powell, M. J., and Ames, A. D. (2014). Planar multi-contact bipedal walking using hybrid zero dynamics. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE.

Lai-Fook, K. and Ambrose, R. (1997). Automation of bioregenerative habitats for space environments. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2471–2476 vol.3.

Lovchik, C. and Diftler, M. (1999). The robonaut hand: a dexterous robot hand for space. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 907–912 vol.2.

Meeussen, W. (2013). ros_control.

Mehling, J. S., Strawser, P., Bridgwater, L., Verdeyen, W., and Rovekamp, R. (2007). Centaur: Nasa's mobile humanoid designed for field work. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2928–2933.

Meredith, M. and Maddock, S. (2004). Real-time inverse kinematics: The return of the jacobian. Technical report, Department of Computer Science, University of Sheffield.

Merhav, N., Gutman, M., and Ziv, J. (1989). On the estimation of the order of a markov chain and universal data compression. *Information Theory, IEEE Transactions on*, 35(5):1014–1019.

Mistry, M., Nakanishi, J., Cheng, G., and Schaal, S. (2008). Inverse kinematics with floating base and constraints for full body humanoid robot control. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 22–27. Look into control techniques How does using velocity affect the base coordinates? Since it is not iterative, maybe incorporating the constraints allows for the true orientation to work its way in.

Morris, B., Powell, M. J., and Ames, A. D. (2013). Sufficient conditions for the lipschitz continuity of qp-based multi-objective control of humanoid robots. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 2920–2926. IEEE.

Nakamura, Y. (1991). *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley.

Office of Management and Budget (2013). Impacts and costs of the october 2013 federal government shutdown. http://www.whitehouse.gov/sites/default/files/omb/reports/impacts-and-costs-of-october-2013-federal-government-shutdown-report.pdf.

Paine, N., Mehling, J. S., Holley, J., Radford, N., Johnson, G., Fok, C., and Sentis, L. (2014). Actuator Control for the NASA-JSC Valkyrie Humanoid Robot: A Decoupled Dynamics Approach for Torque Control of Series Elastic Robots. *Journal of Field Robotics (Submitted 2014)*.

Platt Jr, R., Abdallah, M. E., and Wampler, C. W. (2010). Multi-priority cartesian impedance control. In *Robotics: Science and Systems*.

Pratt, J., Carff, J., Drakunov, S., and Goswami, A. (2006). Capture point: A step toward humanoid push recovery. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 200–207. IEEE.

Rea, R., Beck, C., Rovekamp, R., Diftler, M., and Neuhaus, P. (2013). X1: A robotic exoskeleton for in-space countermeasures and dynamometry. In *AIAA SPACE 2013 Conference and Exposition*.

Said, A. and Pearlman, W. (1996). A new fast and efficient image codec based on set partitioning in hierarchical trees. In *IEEE Transactions on Circuits and Systems for Video Technology*, pages 243–250. IEEE.

Schnabel, R., Wessel, R., Wahl, R., and Klein, R. (2008). Shape recognition in 3d point-clouds. In Skala, V., editor, *The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008*. UNION Agency-Science Press.

Sentis, L. and Khatib, O. (2005a). Control of free-floating humanoid robots through task prioritization. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation. ICRA 2005.*, pages 1718–1723.

Sentis, L. and Khatib, O. (2005b). Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, pages 505–518.

Sentis, L., Park, J., and Khatib, O. (2010). Compliant control of multicontact and center-of-mass behaviors in humanoid robots. *IEEE Transactions on Robotics*, 26(3):483–501.

Sentis, L., Petersen, J., and Philippsen, R. (2013). Implementation and stability analysis of prioritized whole-body compliant controllers on a wheeled humanoid robot in uneven terrains. *Autonomous Robots*, 35(4):301–319.

Sheridan, T. B. (1992). *Telerobotics, Automation, and Human Supervisory Control*. The MIT Press.

Stoytchev, A. (2005). Toward learning the binding affordances of objects: A behavior-grounded approach. In *Proceedings of the AAAI Spring Symposium on Developmental Robotics*, Stanford University.

ten Pas, A. and Platt, R. (2014). Localizing grasp affordances in 3-d points clouds using taubin quadric fitting. In *ICRA*. IEEE.

Tevatia, G. and Schaal, S. (2000). Inverse kinematics for humanoid robots. In *In Proceedings of the International Conference on Robotics and Automation (ICRA2000*, pages 294–299.

Whitney, D. E. (1972). The mathematics of coordinated control of prosthetic arms and manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 94(4):303.

Zhao, Y., Kim, D., Fernandez, B., and Sentis, L. (2013). Phase space planning and robust control for data-driven locomotion behaviors. In *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, Atlanta, Georgia.

Manufacturer	**Name**	**Type**	**Range**	**Resolution**	**FPS**	**Placement**
PMD	Camboard Nano	TOF Depth	0m-1m	160x120	90	Chin, Wrists, Shins
Ensenso	N-10	Texture Aided Stereo	450mm - 1600mm	640x480	8	Forehead
Hokuyo	UTM-30LX-EW	LIDAR	0.1m - 10m	1080 x 1	40	Both Knees
Ibeo	LUX 8L	LIDAR	200m	880x8	25	Eyeball level
Point Grey Research	Flea 3 (FS3-U3-13E4M-C)	Monochromatic Camera	N/A	1280x1024	60 FPS	Left and Right Waist, Right Lower Back
Senix	TSPC-30S1-232	Sonar	4.3m	N/A	20	Waist level

Table 1: Valkyrie Exteroceptive Sensor Suite

Manufacturer	**Name**	**Type**	**Placement**
Invensense	MPU6000	3-axis Gyro and 3-Axis Accelerometer	Both hands and feet
Takktile	Takktile	Pressure Sensor	Fingers and Palm
ATI	9105-NETOEM	6-axis load cell	Both Ankles
Tekscan	9811	Pressure Sensor	Soles of both feet
Futek	TFF60	Linear Load cell	In both linkages of an ankle actuator
Microstrain	3DM-GX3-15	6-axis IMU	Left and Right Shoulder
Microstrain	3DM-GX3-45	9-axis IMU	Top of Skull

Table 2: Valkyrie Proprioceptive Sensor Suite

Constraint Type	**Description**
FlatContact	Constrains all 6 DOFs at a contact point
PointContact	Constrains the 3 lateral DOFs at a contact point
Transmission	Makes the position of a slave DOF dependent on that of a master DOF

Table 3: The Constraint Library

Task Type	**Description**
JPos	Controls every joint's position
CartPos	Controls the Cartesian position of a robot link
QuaternionOrientation	Controls the orientation of a robot link
Wrench	Applies a force and torque to a point on the robot
COM	Controls the position of the robot's COM

Table 4: The Task Library