Possible Exoskeleton Control Architectures and Algorithms
(Draft 4.0)

Alex Ansari, Christopher G. Atkeson, Howie Choset, and Matthew Travers
Carnegie Mellon University
September 30, 2015

1 Executive Summary

1. We recommend using online optimization. This architecture allows a wide variety of control approaches to be implemented and work together.

2. We also recommend a red team that explores traditional frequency domain control system design.

3. We recommend applying as many sensors as possible, and asking the performers to make it easy to add more by making the sensor network available in the design. This maximizes the probability of success with respect to control architectures and algorithms by enabling multiple control approaches to be implemented, refined, and support each other.

4. We recommend a “symbiotic” control system design should be used as a backup, in case more aggressive design philosophies such as “invisible” and “natural” control system designs are not achievable. The “symbiotic” control system design expects the operator to adapt to the exoskeleton and its control, and has the control customized for the operator.

2 Clickable web links

The web links (URLs) below can be clicked on to view, if your PDF viewer/browser supports that. On our browsers you actually have to click on the black part of a letter, not just in the blue box (if present).

3 Scope: What is this paper about?

This paper surveys possible exoskeleton control architectures and algorithms. One goal is to address the question: How do exoskeleton control approaches compare with approaches
to humanoid robot control? A companion paper surveys implemented exoskeleton control architectures and algorithms.


The focus is on exoskeleton control that allows a highly trained and top percentile athletic operator to carry a payload that weighs approximately the same amount as the operator. We envisage these types of exoskeletons to be useful in carrying protective and safety equipment for SWAT teams, police, firefighters, and soldiers.

We expect each exoskeleton controller to be used by and optimized for a single operator. A substantial investment in capturing the operators normal behavior, operator training and learning, and controller customization can be made.

We focus this survey on exoskeleton control for lower body tasks (standing, walking, running, jumping, kicking, dodging, ...). We do not survey exoskeleton control for human arms or manipulation.

We treat the torso and helmet of the exoskeleton as the payload, and focus on a lower body exoskeleton to support any payloads that are on the torso or head.

A future white paper could discuss possible combinations of response to user and exoskeleton autonomy including balance control; response to trips, slips, stumbles, and fumbles; task assistance and guidance (guide operator to doorknob, button, or light switch); superhuman response to external perturbations such as projectiles and explosions; and autonomous execution (operator/exoskeleton symbiosis with multi-tasking)?

A future white paper could discuss how to handle exoskeleton malfunctions and damage to the system. Basically this boils down to fault detection, and switch over to an appropriate safe mode of operation.

A future white paper could evaluate using simulation to assess the various control approaches’ robustness to modeling error and unmodeled dynamics.

A future white paper could survey human learning with respect to exoskeletons and teleoperation.


A future white paper could survey adaptive control of exoskeletons.

The relationship between physical human-exoskeleton interaction and dynamic factors: using a learning approach for control applications

http://info.scichina.com/sciFe/EN/abstract/abstract516165.shtml

A future white paper could survey the role of phase estimation, behavioral clocks, phase reset events such as impacts, central pattern generators, and oscillator approaches in robot and exoskeleton control.
3.1 Related Surveys

Our copy:

3.2 Bias

If you ask people who research complex computationally expensive high end control systems for a living how to control a system, they will recommend complex computationally expensive high end control systems. It is entirely possible that simple local low dimensional controllers designed using traditional (frequency domain) techniques will do quite well, especially if the appropriate sensors (such as operator-exoskeleton force sensors) are available. It may even be the case that frequency domain techniques will handle structural modeling error (jello-like human operators, deformable materials (straps and pads, wires and hoses), and structural deformation and joint play in lightweight frames) better than the state space oriented and other advanced techniques advocated here.

We recommend a red team that explores traditional frequency domain control system design, as well as an emphasis on advanced control approaches.

3.3 Let’s keep it simple

Terms like impedance and admittance control are used, but are often confusing, as in controller design one can choose from several possible inputs into the exoskeleton (exoskeleton positions, velocities, accelerations, operator-exoskeleton contact forces and exoskeleton-world contact forces), and choose from several possible exoskeleton outputs: exoskeleton actuator forces and torques, exoskeleton motion (position, velocity, and acceleration), as well as operator-exo contact forces or exo-world contact forces.

A useful background paper on impedance and admittance:

Variants of nonlinear feedback control such as feedback linearization or sliding mode control are largely ignored in this paper. Once the decision to use feedback control based on a set of observable quantities and with particular outputs is made, one can try out the various linear and nonlinear feedback control paradigms to see what works well.

Variants of function approximation methods such as lookup tables, fuzzy logic, sigmoidal neural networks, radial basis functions, and locally weighted regression are also largely ignored. Once the decision to use a function approximator and what the inputs and outputs are has been made, one can try out the various approaches to see what works well.

Same for optimization methods.

Same for constraint enforcement (such as avoiding self-collisions) in either optimization or feedback control (barrier Lyapunov functions (BLF)).

Stability proofs of any of these methods should be viewed skeptically due to the unmodeled operator-exoskeleton and exoskeleton-world contact dynamics, actuator unmodeled
dynamics, joint play and exoskeleton structural deformation, and controller time delays which are typically ignored, especially in proofs involving passivity arguments or Lyapunov functions. We ignore stability proofs as well.

4 What are some design philosophy alternatives?

4.1 “Invisible” exoskeleton

Can we build an exoskeleton that allows the operator to behave naturally and exerts little or no force on the operator?

This design goal could be achieved through active control using feedback of the operator-exoskeleton forces and/or high quality prediction of what the operator will do next. Additional gravity, friction, actuator, and inertial compensation (inverse dynamics) can improve performance.

4.2 “Natural” exoskeleton

Can we build an exoskeleton that allows the operator to behave naturally? The operator can feel the exoskeleton, but can overpower it when necessary.

One way to achieve this design goal is to have extremely light attachments to the limbs, and put all heavy exoskeleton components on the torso and head. The limbs can be physically driven by the operators muscles with little “drag” on the operator. Bypass valves and clutches could allow the operator to physically “take over” and push the relatively lightweight limb mounted parts of the exoskeleton around.

A complementary way to achieve this goal is through active control. Low impedance (torque source) actuation makes combining active control with operator backdriving easier.

A “natural” exoskeleton is easier to build than an “invisible” exoskeleton.

4.3 “Symbiotic” exoskeleton

Can we build an exoskeleton that allows the operator/exoskeleton team to be effective, but not necessarily behave naturally? The operator needs to learn to “fly” the suit, just as operators learn to operate parachutes, wing suits, diving equipment, high and low temperature protective suits, firefighting equipment, and high altitude flight and parachute suits.

An advantage of this approach is that the exoskeleton can adapt its behavior to the task. For example, for jumping, the exoskeleton can operate like a pogo stick: [https://www.youtube.com/watch?v=1bp41vWP4o4](https://www.youtube.com/watch?v=1bp41vWP4o4) and for running it could act like jumping stilts: [https://www.youtube.com/watch?v=9Z0d7yEyhwI](https://www.youtube.com/watch?v=9Z0d7yEyhwI)

A “symbiotic” exoskeleton is easier to build than an “invisible” or “natural” exoskeleton.
5 Effects of underlying exoskeleton actuator technology

One basic distinction is whether the actuators are thought of as position or force sources. Often this is revealed by the design of the “low level” controller. Note that this is often somewhat confusing. Electric motors are torque sources (low impedance), but when a high gear ratio transmission is added (such as a harmonic drive) the whole system becomes high impedance and thus is best thought of as a position (or velocity) source. Therefore, position or velocity control is performed by the low level controller. Hydraulic actuators are high impedance, but when a force or load sensor is added (or piston differential oil pressure is used as a force sensor), the whole system becomes low impedance and is best thought of as a force source. Force (or joint torque) control is performed by the low level controller. Force or torque sensing can be added to an electromechanical drive train (electric motor plus gears or electric motor plus ballscrew) to make it lower impedance and more like a force source as well.

6 Recommended Control Architecture

We recommend the following hierarchical control architecture.

6.1 Low Level Control (LLC)

Low level control presents an API (Application Programmer Interface) for higher control levels to access the actuators, sensors, and other robot hardware. It implements high servo rate processing including feedback control and online optimization. This level provides an idealization of the exoskeleton in two ways: What degrees of freedom (DOF) are controlled?, and How is a degree of freedom controlled?

6.1.1 LLC: What degrees of freedom are controlled?

The low level control API may directly map on to the exoskeleton hardware (physical joints, raw actuators, and raw sensors), or it may present an “abstracted” view or “virtual” hardware. For example, it is often useful to create virtual degrees of freedom that are actually combinations of joints, or to control points on the robot in Cartesian or other coordinates. We refer to these virtual degrees of freedom as synergies. A complex synergy is the center of mass. Higher level control is made easier by providing a virtual sensor that measures center of mass location and velocity, as well as providing a virtual actuator that acts at the center of mass in a Cartesian fashion (push horizontally and vertically in world coordinates). On a smaller scale, it may simplify higher level control to create virtual actuators that cross multiple joints. These multi-joint actuators are actually implemented by commands to individual physical single-joint actuators.
6.1.2 LLC: How is a degree of freedom controlled?

It is useful to provide human programmers with idealized abstractions of control of a single degree of freedom. Here are some common alternatives that could be provided by low level control:

- Position control. The joint moves to a specified position and stays there independent of load.

- Velocity control. The joint moves at a specified velocity independent of load.

- Force or torque control. The joint provides a specified force or torque independent of its motion.

- Impedance control: The user specifies a desired position (for a spring), and commands a desired stiffness, damping, and potentially a modified inertia or moment of inertia. Modifying inertia is more dangerous than providing variable active stiffness or damping. Or (almost equivalently), the user specifies a desired force or torque which changes according to the position, velocity, and acceleration of the joint.

- Valve control (for fluidic systems)

\[ u = K_q(q - q_d) + K_\dot{q}(\dot{q} - \dot{q}_d) + K_\tau(\tau - \tau_d) + K_{\dot{\tau}}(\dot{\tau} - \dot{\tau}_d) + u_{ff} \]  

where \( u \) is a valve command vector (with a valve command for each actuator), \( q \) is the vector of joint angles, \( \dot{q} \) are joint velocities, \( \tau \) are actuator forces or joint torques, and \( \dot{\tau} \) is the rate of change of the forces or torques (the time derivative of \( \tau \)). The subscript \( d \) indicates a desired value, \( K_q \) is a joint position gain matrix, \( K_\dot{q} \) is a joint velocity gain matrix, \( K_\tau \) is a joint force/torque gain matrix, and \( K_{\dot{\tau}} \) is a joint force/torque derivative gain matrix. \( u_{ff} \) are feedforward valve commands. This formulation takes into account the most important hydraulic actuator dynamics, and allows full state feedback (including \( \tau \) as part of the state), pole placement, and linear quadratic regulator (LQR) control of a third order model of actuator plus rigid body dynamics.

If electric motor and power amplifier dynamics are taken into account in electric motor systems, there would be an analogous equation for the commanded motor voltage of an analog power amplifier or the control input of a PWM amplifier. The torque error term \( K_\tau(\tau - \tau_d) \) could be replaced or augmented by a servo on piston differential pressure in a fluidic system or an electrical current servo in an electric system.

Often this type of control is referred to as a “torque control inner loop” with a “position/velocity control outer loop”. In this approach the inner and outer loops are combined in one equation (eq. [1]), and there is no longer “inner” and “outer” control quantities. All feedback is treated the same way.

- Online optimization of accelerations, actuator forces and torques, and contact forces using quadratic programming. See Section 20 for further explanation.
6.1.3 LLC: Feedback control

PID or more complex nonlinear feedback control laws such as sliding mode control are implemented by low level control. Note that a PID control law can be made (smoothly) nonlinear by changing the PID gains on each servo tick. Distributed low level control with limited communication typically controls each joint or each set of joints (a limb, for example) independently. Centralized low level control can support cross coupling of all joints, and implement “modern” control schemes such as full state feedback, pole placement, linear quadratic regulator (LQR) design, linear quadratic Gaussian (LQG) design, and linear quadratic Gaussian loop transfer recovery (LQG-LTR) design.

6.1.4 LLC: Feedforward control

Gravity, friction, actuator dynamics, and other feedforward disturbance compensation are implemented by low level control, as well as any inverse dynamics or other types of feedforward reference tracking control.

6.1.5 LLC: Existing examples

Typically low level control is implemented by a high sampling rate global servo [Boston Dynamics Atlas humanoid] or distributed limb or joint-level high sampling rate servos [Sarcos Primus humanoid]. Often, this controller is provided by the manufacturer of the robot or exoskeleton and runs on a separate computer system from the “user’s” or task/application controller.

6.2 “Phoneme” Control

This level often provides the behavioral primitives or “verbs” for robot programming: providing mid-level controllers for relatively constant phases of behavior. These behaviors often involve position or velocity targets (goto or stay-at X), and trajectory references. Example phoneme behaviors from walking: left-stance, double-support-1, right-stance, double-support-2, .... Example behaviors from running: left-foot-down, flight-1, right-foot-down, flight-2, .... Example behaviors from horizontal jumping: hjump-pushoff, hjump-flight, hjump-prepare-landing, hjump-impact, hjump-balance, .... Finer or multiple behavioral resolutions may be needed. For example a walking stance period may be broken up into heel-strike, first-half, second-half, pushoff, and toe-off. The first and second halves of stance are divided by when the center of mass passes over the ankle. Local inverse kinematics (finding small joint motions for small target changes) is performed at this level.

6.3 “Word” Control

This level determines the control flow of a behavior: what happens next? Mid-level controllers are provided for sequences of “phoneme” behavioral units. This level often involves finite state machines, timers, if-then condition testing, and gain or behavior scheduling. Example word behaviors: walk-to-location(x=7.8m, y=2.3m), run-to-location(speed=3.7m/s),
and horizontal-jump (height=2m, length=5m). Global inverse kinematics (selection of solution branch) is performed at this level.

6.4 “Sentence” Control: Behavior Selection

The highest level of control in the recommended architecture is behavior selection, which answers the question: “What should I do next?” This level selects behaviors and behavioral parameters (targets, speeds, durations, etc.). This level often also includes decision tables, finite state machines, timers, if-then condition testing, ... This level often involves a great deal of human operator online interaction, or is completely done by the operator.

6.5 State Estimation

In this architecture, state estimation to deal with sensor noise, disturbance estimation, and incomplete or redundant sensing is a separate process and is as independent as possible from the control method. This makes evaluating different control methods much easier. Disturbance observers for disturbances such as moving support surfaces, contact or wind forces, and model errors are also implemented.

6.6 Error Handling

Errors must be detected and handled by parameter (target) adjustment and/or behavior switching at all of these levels.

6.7 Why do we recommend this architecture?

The recommended control architecture is intuitive, abstracts the hardware and lower level control in a straightforward way, is easy to program, and is commonly used. Variants of this architecture were used by almost all teams in the DARPA Robotics Challenge.

6.8 What are the alternatives?

Closely related control architectures combine Phoneme and Word control, or combine Phoneme, Word, and Behavior Selection. Neither of these is a major change.

An alternative behavioral architecture might not have explicit levels but instead has a “soup” of behaviors that compete or cooperate for the ability to control the robot or exoskeleton. Subsumption architecture and other behavior-based architectures are closer to the “soup” approach and to some extent use inhibition and excitation among behaviors to select active behaviors.

Another alternative behavioral architecture does not have explicit separate behaviors, but one or a small number of functions or policies. We are seeing more of these architectures due to the success and popularity of function approximation such as deep learning.

An alternative approach to state estimation is to design state estimation and control laws simultaneously. Another alternative is to have a controller with internal state, and design or learn a policy that works well, but does not have an explicit state estimation part.
7 Operator and exoskeleton dynamics

Exoskeletons are similar to robots, but there are several important differences. There are two sets of contact forces: operator-exoskeleton (which we denote with a subscript ox as in $f_{ox}$) and exoskeleton-world (which we denote with a subscript xw as in $f_{xw}$). If the operator is tightly strapped or rigidly held by other types of physical constraints, we can lump the operator body parts with the exoskeleton parts which they are attached to in terms of link inertias, locations of center of mass, and moments of inertia, and just think of the whole system as one robot (we indicate this with the subscript ox on inertial, Coriolis and centripetal, and gravitational (ICCG) terms in the dynamics equations as in $M_{ox}$). We can also separate out the operator dynamics (indicated by the subscript o on ICCG terms as in $M_{o}$) from the exoskeleton dynamics (indicated by the subscript x on ICCG terms as in $M_{x}$) if this is more convenient (we separate them out and ignore operator dynamics in what immediately follows). If the operator is loosely connected to the exoskeleton, this introduces a huge modeling and control problem, and would also be very dangerous for the operator (as riding a bull or bucking horse in a rodeo is dangerous to a rider). We do not address the loosely connected case in this paper.

The dynamics of an operator standing on the ground (or on parts of the exoskeleton that are on the ground):

$$M_{o}(q)\ddot{q} + C_{o}(q, \dot{q}) + G_{o}(q) + J_{ow}^{T}(q)f_{ow} = \tau_{o} - J_{ox}^{T}(q)f_{ox} \tag{2}$$

where $q$ are the operator joint angles, $\dot{q}$ and $\ddot{q}$ are the corresponding joint velocities and accelerations, $M_{o}$ is the operator inertia matrix, $C_{o}()$ are the operator Coriolis and centripetal forces, $G_{o}()$ are the operator gravitational forces, $J_{ow}^{T}(q)f_{ow}$ (Jacobian matrix multiplied with the contact forces) are the operator-world contact forces expressed as exoskeleton joint torques, $\tau$ are the operator joint torques due to muscle activity and passive tissue mechanics, and $J_{ox}^{T}(q)f_{ox}$ are the operator-exoskeleton forces expressed as operator joint torques. We assume the operator joint friction is negligible.

We assume the exoskeleton joints match some or all of the human joints. The dynamics of such an exoskeleton standing on the ground or attached to the human are:

$$M_{x}(q)\ddot{q} + C_{x}(q, \dot{q}) + G_{x}(q) + F_{x}(q, \dot{q}) + J_{xw}^{T}(q)f_{xw} = \tau_{x} + J_{ox}^{T}(q)f_{ox} \tag{3}$$

$M_{x}$ is the exoskeleton inertia matrix, $C_{x}()$ are the exoskeleton Coriolis and centripetal forces, $G_{x}()$ are the exoskeleton gravitational forces, $F_{x}()$ are the exoskeleton friction forces, $J_{xw}^{T}(q)f_{xw}$ (Jacobian matrix multiplied with the contact forces) are the exoskeleton-world contact forces expressed as operator joint torques, $\tau_{x}$ are the exoskeleton joint torques, and $J_{ox}^{T}(q)f_{ox}$ are the operator-exoskeleton forces expressed as operator joint torques.

The dynamics of the combined system when the operator is tightly strapped in or otherwise “rigidly” attached to the exoskeleton are:

$$M_{ox}(q)\ddot{q} + C_{ox}(q, \dot{q}) + G_{ox}(q) + F_{x}(q, \dot{q}) + J_{ow}^{T}(q)f_{ow} + J_{xw}^{T}(q)f_{xw} = \tau_{o} + \tau_{x} \tag{4}$$

$M_{ox}$ is the combined operator-exoskeleton inertia matrix, $C_{ox}()$ are the combined Coriolis and centripetal forces, and $G_{ox}()$ are the combined gravitational forces. Note that the forces
between the operator and the exoskeleton do not appear explicitly in the combined dynamics, but they can be computed using either equation 2 or 3.

We are ignoring actuator dynamics, which we can handle by augmenting the state vector and adding additional equations.

We do not describe how to derive these equations or identify appropriate model parameters. We can provide this information if requested. We are assuming the equations have been derived taking into account constraints such as one or two feet on the ground. Until we discuss “floating body” dynamics in Section 19, we are assuming contacts do not slip and are not broken. Therefore foot touchdown and liftoff require special handling by controllers using these dynamics, which we will not describe in this paper. We are assuming all degrees of freedom (joints) are actuated by the operator, but not necessarily by the exoskeleton, until we get to “floating body” dynamics (Section 19).

To understand these equations better, it is useful to have all Jacobians equal to the identity matrix. This means the operator is directly applying torques at the exoskeleton joints \((\tau_{\text{ox}})\) (a realistic assumption), and so is the world \((\tau_{\text{ow}} \text{ and } \tau_{\text{xw}})\) (a very unrealistic assumption). In this case, the operator dynamics are:

\[
M_0(q)\ddot{q} + C_0(q, \dot{q}) + G_0(q) + \tau_{\text{ow}} = \tau_{\text{o}} - \tau_{\text{ox}} \tag{5}
\]

the exoskeleton dynamics are:

\[
M_x(q)\ddot{q} + C_x(q, \dot{q}) + G_x(q) + F_x(q, \dot{q}) + \tau_{\text{xw}} = \tau_{\text{x}} + \tau_{\text{ox}} \tag{6}
\]

and the combined dynamics are:

\[
M_{\text{ox}}(q)\ddot{q} + C_{\text{ox}}(q, \dot{q}) + G_{\text{ox}}(q) + F_x(q, \dot{q}) + \tau_{\text{ow}} + \tau_{\text{xw}} = \tau_{\text{o}} + \tau_{\text{x}} \tag{7}
\]

8 What do we want from intent prediction?

For many of the following control systems, no operator intent estimation or prediction is needed. When we get to inverse dynamics, we need to know a desired acceleration \(\ddot{q}_d\). In addition, operator force estimates \((\hat{f}_o \text{ or } \hat{\tau}_o)\) or operator-exoskeleton force estimates \((\hat{f}_{\text{ox}} \text{ or } \hat{\tau}_{\text{ox}})\) are useful. For online optimization, the more we know about the future, the better online optimization can perform. We discuss this in Section 20.1.

9 Passive control with no active control

One control option is to use only passive controls on a joint or on all joints, and no active controls or actuators. Such a system would rely on counterbalancing, mechanical springs, air springs, etc. to reduce operator forces. An operator would rely on mechanical backdrivability to drive exoskeleton to do tasks. These passive devices have no controls, and can be included in the dynamics equations. In this case, \(\tau_{\text{x}}\) is zero for any passive degree of freedom.
10 Open loop control

Another control option is open loop control, in which actuator commands are constant, or a function of time or phase of a behavior or task (at least on the small time scale). Commands or behaviors may be selected at a relatively coarse time interval by the operator or by the exoskeleton controller. This approach typically uses impedance control in either joint or Cartesian coordinates with either pre-generated references, selected targets, or online generated trajectories to move to targets or perform tasks. The operator relies on mechanical backdrivability to correct exoskeleton behavior. In this case, it is common to generate the exoskeleton actuator commands using a time or phase index: \( \tau_x = \tau_{ff}(t) \) where \( \tau_{ff} \) is a time or behavioral phase dependent feedforward torque vector.

11 Fixed feedforward control with feedback control

We can add our favorite feedback control scheme to open loop control. Now the control is closed loop, but the feedforward control to achieve a trajectory is designed and executed independently of any feedback control. In most practical robotics manually designed fixed gain independent joint linear (PID) feedback control is used. As previously mentioned, more complex feedback control approaches such as approaches that coordinate across joints (full state feedback, pole placement, linear quadratic regulator (LQR) design, linear quadratic Gaussian (LQG) design, and linear quadratic Gaussian loop transfer recovery (LQG-LTR) design) or nonlinear alternatives that handle discontinuous control problems such as sliding mode control can also be used. Feedback gains can be “gain scheduled”, which means that feedback gains are pre-tabulated and looked up and deployed based on the current position, velocity, or some other quantity (temperature, for example).

12 Active gravity compensation

Now we will examine feedforward control that varies and is computed based on the current state of the exoskeleton. Active exoskeleton gravity compensation is similar to counterbalancing or physical gravity compensation, except it is done with exoskeleton actuators to cancel the exoskeleton weight (but not inertia):

\[
\tau_x = \hat{G}_x(\hat{q})
\]  

(8)

where \( \tau_x \) are the exoskeleton joint torques, \( \hat{G}_x() \) are the exoskeleton gravitational forces calculated using an imperfect model, and \( \hat{q} \) is the estimated configuration of the operator (and exoskeleton). The operator relies on mechanical backdrivability to drive the exoskeleton to do tasks. The weight of the operator can also be supported by the exoskeleton in full gravity compensation:

\[
\tau_x = \hat{G}_{\text{tot}}(\hat{q})
\]  

(9)

where \( \hat{G}_{\text{tot}}() \) are the combined operator and exoskeleton gravitational forces calculated using an imperfect model,
Sarcos Primus Humanoid videos of active gravity compensation applied to a humanoid robot:
CMU: cancel gravity and friction: https://www.youtube.com/watch?v=Ac5cowTw0Pw
ATR: cancel gravity only: https://www.youtube.com/watch?v=KNxxLm4sPys

A simple approximation for a lower body exoskeleton with a heavy payload at the torso is:

\[ \tau_x \approx \hat{J}^T(\hat{q})(\hat{m}\hat{g}) \]  

where \( \hat{J} \) is the estimated Jacobian matrix for the center of mass, \( \hat{m} \) is the estimated weight of the exoskeleton (which could include the operator), and \( \hat{g} \) is the gravity vector (it is estimated due to possible orientation errors in the controller’s estimate of vertical). All weight is assumed to be concentrated at the center of mass of the full system.

13 Friction compensation

Friction can also be compensated using feedforward control.

\[ \tau_x = \hat{F}_x(\hat{q}, \dot{\hat{q}}) \]  

are the estimated friction torques at the estimated exoskeleton configuration and velocity. Note that friction may be load dependent, and may require additional state to correctly handle stiction and hysteresis effects.

Sarcos Primus Humanoid video of friction compensation applied to a humanoid robot:
CMU: cancel gravity and friction: https://www.youtube.com/watch?v=Ac5cowTw0Pw

14 Online inverse dynamics control

The goal here is to map from desired motion (in this case desired accelerations) to exoskeleton actuator commands (and desired operator and world contact forces). The exoskeleton actuator torques attempt to match the torques caused by exoskeleton inertial, Coriolis, centripetal, gravitational, and frictional forces.

\[ \tau_x = \tau_{x,ID} = \hat{M}_x(\hat{q})\ddot{\hat{q}}_d + \hat{C}_x(\hat{q}, \dot{\hat{q}}) + \hat{G}_x(\hat{q}) + \hat{F}_x(\hat{q}, \dot{\hat{q}}) \]  

If in addition we want to cancel exoskeleton-world contact forces, we add another term:

\[ \tau_x = \tau_{x,ID+Fw} = \tau_{x,ID} + \hat{J}^T_{sw}(\hat{q})\hat{f}_{sw} \]  

If there are actuator dynamics or other dynamics, we can include those dynamics as well.

These force/torque estimates rely on estimated dynamic models and their parameters. If the models and estimated state (position and velocity) are perfect, we get perfect cancellation of most of the dynamics:

\[ M_x(q)(\ddot{q} - \ddot{q}_d) = J^T_{ox}(q)f_{ox} \]  

12
If $\ddot{q}_d$ is set to zero, the operator directly drives the acceleration of the exoskeleton with an effective inertia of $(M_x^{-1}(q)J_{ox}^T(q))^{-1}$

$$\ddot{q} = M_x^{-1}(q)J_{ox}^T(q)f_{ox}$$  \hspace{1cm} (15)$$

$\ddot{q}_d$ can be used to drive the exoskeleton along trajectories, and the operator can directly modify the current trajectory.

Simplifying equation 15 by assuming the operator directly applies torques at the exoskeleton joints we get:

$$M_x(q)\ddot{q} = \tau_{ox}$$  \hspace{1cm} (16)$$

The operator sees the true inertia of the exoskeleton, but no gravitational loads and no other dynamics (Coriolis, centripetal, or frictional).

Note that we cannot change the apparent inertia of the exoskeleton without some form of acceleration or force feedback. With acceleration feedback we can make the exoskeleton have a different inertia. Setting $\ddot{q}_d$ to zero and adding acceleration feedback:

$$\tau_x = \tau_{x-ID+Fw} - (M_d(\ddot{q}) - \dot{M}_x(\dddot{q}))\dddot{q}$$  \hspace{1cm} (17)$$

sets the apparent exoskeleton inertia to $M_d$ if the dynamic models are perfect, and state estimation produces not only perfect positions and velocities but also perfect acceleration estimates. In the case where the operator directly applies torques at the exoskeleton joints we get:

$$M_d(q)\ddot{q} = \tau_{ox}$$  \hspace{1cm} (18)$$

In robotics this is usually considered a bad idea if $M_d$ is smaller (a complex issue for a matrix) than $M_x$ at any $q$. This type of control is vulnerable to unmodeled dynamics. However, in an exoskeleton, since we have not tried to cancel the human’s dynamics, the human stabilizes a possibly unstable exoskeleton (if the straps are tight enough).

With force feedback we can also make the exoskeleton have a different inertia. Setting $\ddot{q}_d$ to zero and adding force feedback:

$$\tau_x = \tau_{x-ID+Fw} + K\dddot{f}_{ox}$$  \hspace{1cm} (19)$$

In the case where the force measurements are perfect and the operator directly applies torques at the exoskeleton joints we get:

$$(K + I)^{-1}M_x(q)\ddot{q} = \tau_{ox}$$  \hspace{1cm} (20)$$

By setting $K = M_d^{-1}(q)M_x(q) - I$ we can achieve

$$M_d(q)\ddot{q} = \tau_{ox}$$  \hspace{1cm} (21)$$

Online inverse dynamics control is a generalization of active gravity compensation to include inertial, Coriolis, and centripetal forces (forces due to acceleration and velocity), and potentially frictional forces. Online inverse dynamics can be added to open loop behavior execution and many other types of control such as operator force feedback control, impedance control, and admittance control. Computed torque control and feedback linearization are forms of online inverse dynamics.

A key question is where does the desired acceleration $\ddot{q}_d$ come from? This is discussed in a companion paper. [http://www.cs.cmu.edu/~cga/exo/intent.pdf](http://www.cs.cmu.edu/~cga/exo/intent.pdf)
15 Operator-Exoskeleton Impedance Control

What is impedance? [https://en.wikipedia.org/wiki/Mechanical_impedance](https://en.wikipedia.org/wiki/Mechanical_impedance) Stiffness, damping, and mass are components of an impedance, as they map kinematic variables (position, velocity, and acceleration) to forces and torques.

The goal here is to make the exoskeleton imitate a desired linear dynamic system from the point of view of the operator. Section [18.3] discusses how to make the exoskeleton imitate a desired linear dynamic system from the point of view of external perturbations. In the case where there is no force sensing between the operator and the exoskeleton, the exoskeleton controller maps from exoskeleton configuration and velocity to desired joint/actuator/synergy forces. In this case the actuators need to generate forces and torques rather than positions or angles, or linear or angular velocities.

Here we make the exoskeleton appear as a locally linear impedance in joint coordinates to the operator. We can also make the exoskeleton appear as a locally linear impedance to the operator in some other coordinate system, such as Cartesian coordinates. A different goal is to make the exoskeleton or combined operator and exoskeleton appear as a locally linear impedance to the outside world in some coordinate system.

To make the exoskeleton appear as a locally linear impedance in joint coordinates to the operator, we add position and velocity feedback to $\tau_x$:

$$\tau_x = \tau_x^{\text{ID+Pw}} - K_q (\dot{q} - \dot{q}_{d}) - K_{\dot{q}} \dot{q}$$

so if the dynamic models and state estimation are perfect we get this impedance:

$$M_{\dot{q}} + K_q \dot{q} + K_{\dot{q}} (q - q_{d}) = \tau_{ox}$$

and the operator sees the desired impedance.

Although impedance control is extensively utilized in rehabilitation robotics [18], only a limited number of studies has been focused on this method for power augmentation [19, 20].


16 Force feedback and get out of the way control (inverse dynamics version)

The goal here is to use force sensing between the operator and the exoskeleton to ultimately generate exoskeleton velocities or angular velocities, imitating a desired linear dynamic sys-
tem. This type of control is often referred to as admittance control as the exoskeleton maps contact forces into joint motion. Inverse dynamics control can be used to improve this type of control performance.

What is admittance? [https://en.wikipedia.org/wiki/Admittance](https://en.wikipedia.org/wiki/Admittance) Compliance, inverse damping, and inverse mass are components of an admittance, as they map force and torques to kinematic variables (position, velocity, and acceleration).

We can implement operator force control if measurements of the operator contact forces with the exoskeleton are available. The exoskeleton moves as to create as little force on the operator as possible assuming the actuators are torque sources:

\[
\tau_x = \tau_{x-ID} + F_w - J_{ox}^T(q)f_{oxd} - J_{ox}^T(q)K_{f_{ox}}(\hat{f}_{ox} - f_{oxd})
\]  

(24)

\(K_{f_{ox}}\) is a gain matrix. \(J_{ox}\) is the Jacobian matrix for the operator force \(f_{ox}\). Note that this does not require inverting the Jacobian matrix. We will see that the position control version does invert the Jacobian matrix.

Applying this to force between the operator and the exoskeleton, and assuming that the operator directly applies exoskeleton joint torques,

\[
\tau_x = \tau_{x-ID} + F_w + \tau_{oxd} - K_{\tau_{ox}}(\hat{\tau}_{ox} - \tau_{oxd})
\]  

(25)

So if our dynamic models and state estimation are perfect, we get:

\[
\ddot{q} = -M_x^{-1}(q)(\tau_{oxd} + K_{\tau_{ox}}(\dot{\tau}_{ox} - \tau_{oxd}))
\]  

(26)

One can add force damping terms, which sometimes help:

\[
\tau = \tau_{x-ID} + F_w + \tau_{oxd} - K_{\tau_{ox}}(\dot{\tau}_{ox} - \tau_{oxd}) - K_{f_{ox}}\hat{\tau}_{ox}
\]  

(27)

Note that impedance control can also be implemented using force control by setting:

\[
\tau_{oxd} = K_q(\dot{q} - q_d) + K_q\dot{q}
\]  

(28)

in addition to directly including the stiffness and damping terms in equation\[22\]

### 17 Mapping External Forces To Operator Forces Using Force Control

So far we have ignored or canceled the effect of the exoskeleton contact with the outside world. A different control objective could be to map the external forces to desired operator forces:

\[
f_{oxd} = F_{ox-xw}(f_{xw})
\]  

(29)

and cancel the external forces not transferred in the inverse dynamics. Everything in the above equation is estimated, so we are dropping the hats.`

For example, we could want the operator to feel a scaled (scale factor \(\alpha\)) version of the external forces, as if they were contacting the operator directly.

\[
f_{oxd} = \alpha J_{ox}^{-T}(q)J_{xw}^T(q)f_{xw}
\]  

(30)
and we scale the \( J^T_{tw}(q)f_{sw} \) term in the inverse dynamics by \((1 - \alpha)\). We are assuming we have any necessary force sensing between the exoskeleton and the world.

If the operator applies joint torques, this simplifies to

\[
\tau_{cxd} = \alpha J^T_{tw}(q)(f_{sw})
\]

and the inversion of a Jacobian matrix is unnecessary.

This would provide a way of enabling the operator to perceive and control external contact forces.

18 Autonomous exoskeleton control

18.1 Autonomous exoskeleton position/velocity control

A control objective could be to have the exoskeleton to move autonomously, which is useful if the operator would like to rest, is attending to some other task, or is disabled.

This is usually done by using stored trajectories or trajectories generated online that specify desired position \( q_d \), desired velocity \( \dot{q}_d \), and desired acceleration \( \ddot{q}_d \). The feedforward method is to compute the needed torques (potentially offline) using inverse dynamics (including the operator link inertias, link centers of mass, and link moments of inertia in the dynamic model) based on the desired position, desired velocity, and desired acceleration. Trajectory tracking errors are compensated for by feedback control designed using some other method. In most robotics fixed gain independent joint linear control is used. This method is safest in the sense that the feedback controller can be tested and verified independently of the desired trajectories and dynamic models.

Computed torque approaches use online inverse dynamics computed from the estimated position, estimated velocity, and desired acceleration. In this case the feedback control to compensate for trajectory tracking errors can be independent of the inverse dynamics and its modeling errors, or the desired acceleration can be modified with feedback terms.

\[
\ddot{q}_d^* = \ddot{q}_d + K^q\ddot{q}(\dot{q} - q_d) + K^q(\dot{q} - \dot{q}_d)
\]

The problem with putting the inverse dynamics in the feedback loop (equation 32) is that the effective gains of this control vary widely with the configuration \( q \). Unmodeled dynamics (that also vary with \( q \)) limit the size of the gain matrices \( K^q \) and \( K^q_\dot{q} \). Because so many quantities are varying quite a bit, constant gain matrices may be quite small for a worst case design. It is also very difficult to evaluate and verify this type of nonlinear feedback controller. For these reasons most current approaches to humanoid robots design the feedback controller separately (and manually) and use constant gain independent joint linear feedback control.

18.2 Autonomous exoskeleton-world force control

A control objective could be to have the exoskeleton to autonomously apply a desired force. One could add

\[
J^T_{tw}(q)f_{swd}
\]
to a gravity compensated control or to control including inverse dynamics.

Active force control could be added to improve the performance:

\[ J^T_{xw}(q)K_{f_{xw}}(f_{xw} - f_{xwd}) \]

### 18.3 Autonomous exoskeleton-world impedance control

Another control objective would be to allow the exoskeleton-world contact to be compliant or have an assigned impedance. Let’s describe a small position offset of the exoskeleton-world contact points as \( q_w \). We know that:

\[ \dot{q}_w = J_{xw}(q)\dot{q} \]

and that

\[ \tau_x = J^T_{xw}(q)f_{xw} \]

This allows us to relate stiffness and damping in the external world coordinates with the stiffness and damping in exoskeleton joint coordinates. Here is the relationship for damping:

\[ \tau_x = J^T_{xw}(q)K_w J_{xw}(q)\dot{q} \]

So the damping in joint coordinates is \( \tau_x = K_q \dot{q} \) and therefore

\[ K_q(q) = J^T_{xw}(q)K_w J_{xw}(q) \]

The mapping is the same for stiffness:

\[ K_q(q) = J^T_{xw}(q)K_w J_{xw}(q) \]

So we should assign the exoskeleton joint position gains to be \( K_q(q) \) and the joint velocity gains to be \( K_q(q) \) in equation 22.

Similarly, we can use exoskeleton-world force control to improve performance, setting

\[ f_{xwd} = -K_w(\dot{q}_w - q_{wd}) - K_{w\dot{q}} \dot{q}_w \]

and add the force error:

\[ \tau_x = \tau_{x-ID} - K_q(\dot{q} - q_d) - K_q \dot{q} + J^T_{xw}(q)K_{f_{xw}}(f_{xw} - f_{xwd}) \]

### 19 Underactuated and overactuated control and floating body dynamics

This section describes how to handle underactuation and overactuation using “floating body dynamics.” We have been sweeping this issue under the rug so far. It is now useful to be more explicit. There are two kinds of underactuation we want to consider. The first is “floating body dynamics”, where the robot is not bolted to the floor. Contacts are made and broken during walking, running, and many other behaviors, and contacts slip. Humanoid
robots (and humans) have floating body dynamics. A “root” location (usually somewhere on the pelvis) is chosen, and the position, velocity, orientation, and angular velocity of that point and the rigid body it is on are included in the state vector \( q \). The corresponding dimensions of the velocity vector \( \dot{q} \) are not actuated. A Jacobian matrix \( J \) is used to map from actuation variables \( u \) to joint torques \( \tau \):

\[
\tau = J u
\]

The rows in \( J \) corresponding to the root linear and angular velocities are all zero. To maintain compatibility in our joint representation, we will similarly map operator “muscle commands” \( u_o \) to operator joint torques \( \tau_o \).

\[
\tau_o = J_o u_o
\]

The second kind of underactuation appears in exoskeletons when we rely on the operator to actuate a particular joint. For example, many lower body exoskeletons only actuate the joints in the sagittal plane, leaving lateral and twist (yaw) joints unactuated or passively actuated. Many “exoskeletons” or orthoses only actuate one joint on each limb (typically the ankle or knee). In these situations the question arises as to what is the scope of control: should the device only consider just the actuated joints, or should it consider a wider range of operator joints? Many of these devices do not have enough sensing (such as an IMU) to even estimate the state of other operator joints. We believe that for high performance the entire state of the operator must be considered in control, and we describe a way to do it in Section 20. However, we are not aware of any system that goes this far in terms of control.

Passive actuation can be handled by augmenting the dynamics equations to include the force equation of the passive element, such as a nonlinear spring.

“Missing joint” underactuation can be handled by reducing the size of the \( u_o \) vector and changing the matrix \( J \) to reflect the missing actuation. We can have redundant actuation (more than one actuator across any particular joint). Here we augment \( u_o \) and \( J \) to accommodate extra actuators. We can have multi-joint actuation, which can be handled in \( u_o \) and \( J \) as well.

The modified dynamics equations are

\[
M_o(q)\ddot{q} + C_o(q, \dot{q}) + G_o(q) + J_{ow}^T(q)f_{ow} = J_o u_o - J_{ox}^T(q)f_{ox}
\]

(44)

\[
M_x(q)\ddot{q} + C_x(q, \dot{q}) + G_x(q) + F_x(q, \dot{q}) + J_{xw}^T(q)f_{xw} = J_x u_x + J_{ox}^T(q)f_{ox}
\]

(45)

\[
M_{ox}(q)\ddot{q} + C_{ox}(q, \dot{q}) + G_{ox}(q) + F_{x}(q, \dot{q}) + J_{ow}^T(q)f_{ow} + J_{xw}^T(q)f_{xw} = J_o u_o + J_x u_x
\]

(46)

Note that \( \ddot{q} \) and the actuation vectors \( u_o \) and \( u_x \) do not have the same dimensionality. What difference does that make?

Underactuated joints reduce our ability to control behaviors directly through the actuators. In some cases we can control unactuated degrees of freedom through indirect effects over time, and in some cases those degrees of freedom are not “controllable” in the technical sense. Remember that we have a human operator that fully actuates the exoskeleton, so having some degrees of freedom that are not controllable by the actuators is not fatal to a design.
The no control case is unchanged with underactuation.

Open loop control and fixed feedforward control with feedback control continue to work with underactuation as long as a drive function can be found to generate the desired behavior.

Gravity compensation and friction compensation still work on the actuated degrees of freedom. Passive actuation or the operator needs to carry the load on the unactuated degrees of freedom.

Full (operator) body online inverse dynamics as described in Section 14 no longer works if \( J_x \) is not invertible (not square or not full rank). In the underactuated and overactuated cases \( J_x \) is not square. For a partial (operator) body exoskeleton we can rewrite the dynamics in terms of only those joints the exoskeleton actuates, and invert those dynamics to cancel exoskeleton effects on those joints. This is what has been done with the BLEEX exoskeleton, for example.

However, online optimization as described in the next Section can be used to perform full (operator) body inverse dynamics by finding desirable combinations of accelerations \( \ddot{q} \), contact forces \( f \), and actuator commands \( u \).

Another approach to solving this problem is prioritized control, which does not include optimization. For reasons we will describe, we believe online optimization is much more flexible and can handle programming exoskeleton behavior much better, so we will not discuss prioritized control further.

20 Using Online Optimization: Quadratic Programming

The dominant paradigm in force controlled (typically hydraulic) full size humanoid robots is to use online optimization to handle under and overactuation and to handle constraints in inverse dynamics such as torque and center of pressure limits. Online optimization typically solves the constrained inverse dynamics problem using quadratic programming. This approach is used to implement and trade off among the many control objectives described so far.

Quadratic programming solves the following problem: Minimize

\[
x^T Q x + c^T x
\]

with inequality constraints

\[
A_1 x \leq b_1
\]

and equality constraints

\[
A_2 x = b_2
\]

The optimization variable \( x \) includes accelerations \( \ddot{q} \), actuator commands \( u \), and contact forces \( f \). We could optimize only taking into account the exoskeleton and its dynamics, minimizing the discrepancy between actual and desired accelerations, actuator command magnitude, and contact force magnitude:

\[
C_x(x) = (q - q_d)^T W_q (q - q_d) + u_x^T W_{u_x} u_x + f_{ox}^T W_{f_{ox}} f_{ox} + f_{fw}^T W_{f_{fw}} f_{fw}
\]
\( W_q, W_{ux}, W_{fox}, \) and \( W_{f_{ox}} \) are weight matrices to trade off the different errors/magnitudes and error/magnitude directions. The dynamics (equation 45) are enforced as an equality constraint. Actuator limits, friction limits, center of pressure location constraints, and unidirectional contact constraints are all enforced as inequality constraints. Variants of this optimization were performed every 1-2ms by various teams to control the Atlas robots in the DARPA Robotics Challenge.

Alternatively, we could optimize the operator-exoskeleton combination, minimizing:

\[
C_{ox}(x) = (q - q_d)^T W_q (q - q_d) + u_x^T W_{ux} u_x + f_{ox}^T W_{f_{ox}} f_{ox} + f_{ow}^T W_{f_{ow}} f_{ow} + f_{ox}^T W_{f_{oxw}} f_{oxw}
\]  

(51)

In this case we would enforce the combined dynamics (equation 46) as an equality constraint.

Note that we assume the operator muscle command \( u_o \) is fixed. We include the operator-exoskeleton force \( f_{ox} \) so it can be minimized or set to a desired value.

These approaches can be used to achieve desired joint or synergy (such as center of mass) accelerations, center of mass dynamic behavior, center of pressure location, weight distribution, maintenance of a desired body part position, orientation, velocity, or trajectory, and joint/actuator position and velocity limits (indirectly).

### 20.1 What does online optimization want from operator intent prediction?

Online optimization for the current instant in time tries to match actual accelerations to desired accelerations, so some estimate of the desired acceleration \( \ddot{q}_d \) is useful. In addition, to accurately estimate the dynamics, we need an estimate of the operator-exoskeleton force \( f_{ox} \), and the operator-world force \( f_{ow} \). These estimates could be deterministic values or probability distributions. Probability distributions are more expensive to optimize.

There are more complex versions of online optimization that consider a future time horizon, and use some form of receding horizon control (RHC or MPC) to optimize a time window on each servo tick. In this case knowledge of future desired accelerations is useful, as well as future operator forces. This knowledge could be in the form of deterministic trajectories, probability distributions of desired accelerations and operator forces at each future time, or probability distributions for trajectories of desired accelerations and operator forces, which could take into account the dependence of future desired accelerations and operator forces on previous accelerations and forces.

### 20.2 Controlling operator forces

In the case of exoskeletons, part of the optimization criterion could be to minimize contact forces on the operator by including a term penalizing those forces:

\[
f_{ox}^T W_{f_{ox}} f_{ox}
\]

(52)

Operator force feedback and impedance control could be implemented by specifying a desired operator force \( f_{oxd} \) and adding a “hard” equality constraint:

\[
f_{ox} = f_{oxd}
\]

(53)
Force damping can be implemented by adding the following “soft” constraint term to the optimization criterion:

$$\begin{align*}
\dot{f}_{ox}^T W_{\dot{f}_{ox}} \dot{f}_{ox}
\end{align*}\) (55)

20.3 Reducing operator internal forces

Underactuated exoskeletons may rely on operator bones and tissue to transmit forces. In this case, there is a risk of tearing tendons, ligaments, and muscle, bruising soft tissue, damaging joints, and fracturing or breaking bones. Online optimization can help minimize forces within the operator’s body. Note that it is not net joint forces (the minimal joint forces that are computed by inverse dynamics) that matter. It is local internal forces, such as forces within a bone or soft tissue. Since internal forces can cancel each other, local internal forces are usually larger than internal forces computed from net joint torques. We also note that humans often coactivate their muscles to increase joint stiffness, and to maintain structural integrity of their bodies under load. A prediction of minimal muscle activation to achieve net joint torques will often underestimate the amount of muscle activation, muscle forces, and the resulting forces in and on bones and soft tissue.

To minimize internal forces, we will create a set of internal “virtual” sensors that “measure” internal forces at key points, resulting in vector of internal forces $\mathbf{b}$. In reality, we will use a neuromuscular model of the individual operator, their previous muscle activations, and predictions of their future muscle activations to calculate $\mathbf{b}$. We can include a penalty on the deviation of $\mathbf{b}$ from a desired value $\mathbf{b}_d$ in online optimization:

$$\begin{align*}
(\mathbf{b} - \mathbf{b}_d)^T W_{\mathbf{b}} (\mathbf{b} - \mathbf{b}_d)
\end{align*}\) (56)

20.4 Controlling perceived exoskeleton impedance

We can control the exoskeleton impedance the operator feels in several ways: We can add the impedance equations as “hard” equality constraints (to achieve them exactly)

$$\begin{align*}
J_{ox}^T f_{ox} = K_q(\dot{q} - q_d) + K_q \dot{q}
\end{align*}\) (57)

or by adding the “soft” constraint “equation error” to the optimization criterion weighted by a weight factor $W_{imp}$ to indicate how large an error is tolerated:

$$\begin{align*}
e = J_{ox}^T f_{ox} - K_q(\dot{q} - q_d) - K_q \dot{q}
\end{align*}\) (58)

and

$$\begin{align*}
e^T W_{imp} e
\end{align*}\) (59)

We can alternatively just set the desired operator contact force to:

$$\begin{align*}
f_{oxd} = J_{ox}^{-1}(K_q(\dot{q} - q_d) + K_q \dot{q})
\end{align*}\) (60)
Unfortunately, this involves inverting the Jacobian matrix, which may introduce problems. Assuming that the operator directly applies exoskeleton joint torques, we can get rid of the Jacobian matrix, and this becomes the equality constraint

\[ \tau_{ox} = K_q(\dot{q} - q_d) + K_q \dot{q} \]  

or a penalty term is added to the optimization criterion:

\[ (\tau_{ox} - \tau_{oxd})^T W_{\tau_{ox}} (\tau_{ox} - \tau_{oxd}) \]  

with

\[ \tau_{oxd} = K_q(\dot{q} - q_d) + K_q \dot{q} \]

Mapping external forces to operator forces can be done either with the “hard” equality constraint:

\[ f_{ox} = F_{ox-xw}(f_{xw}) \]

or using the “soft” constraint term:

\[ (f_{ox} - F_{ox-xw}(f_{xw}))^T W_{f_{map}} (f_{ox} - F_{ox-xw}(f_{xw})) \]

Similar techniques are used to handle autonomous exoskeleton position, velocity, force, and impedance control.

20.5 How can the operator share control with autonomous control?

The use of online optimization makes sharing control with an operator easy. Dimensions (even actuated dimensions) that are under the operator’s control have low weights in the online optimization. It is also easy to map operator controls such as joysticks, sliders, and alternative interfaces to almost any desired control outcome by including these measurements in the online optimization. The operator and the controller can “share” or “blend” control by making their respective weight matrices similar.

Another approach to shared control is allocating time slices or providing assistance on an as-needed basis.


This recent paper discusses shared control
http://www.ri.cmu.edu/pub_files/2015/7/Javdani15Hindsight.pdf

20.6 Full robot implementations.

More details on our work and pointers to other work are in the papers at:

http://www.cs.cmu.edu/~cga/drc

Figure I shows the architecture of our robot control based on hierarchical optimization. It was successfully tested on our SARCOS Primus Humanoid and ATLAS (Boston Dynamics) platforms. Our ATLAS robot demonstrated rough terrain walking, ladder climbing, and full body manipulation in the DARPA Robotics Challenge (DRC) Trials of 2013. This work
also received a “Best Oral Paper Award” at Humanoids 2013. Figure 2 shows a sequence of snapshots of ATLAS walking on rough terrain and plots of measured trajectories of the feet and the center of mass (CoM) for the DRC Trials. Figure 3 shows similar data for ladder climbing. In the DARPA Robotics Challenge Finals in 2015 our robot was the only biped that did not fall down or need to be rescued by humans during the two days of the evaluations. Our robot performance was also the most consistent across the two days. Because of the way the evaluations were run, we have only limited data from the actual evaluations. Videos of our robot’s performance during the DRC Finals are available.

In the controller architecture, the Behavior Library (Fig. 1) contains learned behaviors and behaviors created from demonstration or motion capture of humans or other robots, and also acts as a cache for planning. The output of the library is used to prime, speed up, and guide the real time optimization process, and can be motion or force trajectories in joint or task coordinates ($x_d(t)$), as well as possibly including value, heuristic cost, or utility functions ($V_x(t)$) that can be used to guide search and optimization. Libraries can be built in advance, or when task specification are known. The rest of the controller can also operate without information from the library if suitable information is not available.

The Multistep Low-Order Dynamics Planner uses receding horizon control to continually optimize trajectories of motion and force of a simplified model of the current task. In walking, this is usually center of mass ($COM_d(t)$), angular momentum ($L_d(t)$), contact force ($\text{contacts}_d(t)$) trajectories, and value, heuristic cost, or utility functions ($V_c(t)$) specifying tradeoffs between these quantities. We optimize with a time horizon of several seconds using Differential Dynamic Programming.
Figure 3: ATLAS climbing the top half of the same ladder as used in the DRC. Photo snapshots taken every 13 seconds. The top row shows repositioning of the hook hands, and the bottom row shows stepping up one tread. The two plots show ATLAS climbing the first five treads during the actual run at the DRC Trials. Axis definitions and color codes for foot and CoM trajectories are the same as in Fig. 2. Left and right hand positions are plotted with cyan and magenta dashed lines.

For walking, our planner is similar in spirit to Kajita’s Preview Control for a Linear Inverted Pendulum Model (LIPM). We are also using a CoM model, reason about Zero Moment Point (ZMP), and use future information to guide the current trajectory. However, our planner can be generalized to nonlinear models and adjust foot steps while optimizing the CoM trajectory. Like capture point methods, we take the next few steps into consideration but do not plan to come to rest at the end. Ogura et al. investigated generating human-like walking with heel-strike and toe-off by parameterizing the swing foot trajectory. In contrast, we use very simple rules to guide the low level controller to achieve the same behaviors.

The Full Body QP Optimization uses quadratic programming (QP) with a very short time horizon (1ms on SARCOS and 2ms on ATLAS) to optimize the desired full state of the robot ($\theta_d$, $\dot{\theta}_d$), joint torques ($\tau_d$), and contact forces ($f_d$). Currently, we pre-specify gain matrices ($K_{\theta}$, $K_{\dot{\theta}}$, $K_{\tau}$, $K_f$) for the robot controller, but plan to explore how to optimize gains as well as trajectories. This module enforces constraints such as robot kinematics and dynamics and joint, velocity, torque, actuator, friction cone, and center of pressure limits, and resolves redundancies. The module performs inverse kinematics and dynamics to provide us with compliant motions and dynamic behaviors, while compensating for the effects of modeling errors.

For the QP, we use a formulation developed in our group. Unlike those who use orthogonal decomposition to project the allowable motions into the null space of the constraint Jacobian, and minimize costs in the contact constraints and the commands, we directly optimize a quadratic cost in terms of state accelerations, torques and contact forces on the full robot model. This design choice allows us to trade off physical quantities of interest. We are also able to directly reason about inequality constraints such as center of pressure within the support polygon, friction, and torque limits. Although it becomes a bigger QP problem, we are still able to solve it in real time. Hutter et al. resolved redundancy in inverse dynamics using a kinematic task prioritization approach that ensures lower priority tasks always exist in the null space of higher priority ones. In contrast to their strictly hierarchical approach, we minimize a sum of weighted terms. We can directly specify the relative importance of
the terms by adjusting the weights.

The **Low-Level Robot Controller** is a blend of our work and controllers provided by the robot manufacturers. For SARCOS and ATLAS, hydraulic valve commands for each joint \(i\) are generated by high rate servo controllers provided by the manufacturers,

\[
v_i = k_{\theta,i}(\theta_i - \theta_{d,i}) + k_{\dot{\theta},i}(\dot{\theta}_i - \dot{\theta}_{d,i}) + k_{\tau,i}(\tau_i - \tau_{d,i}) + v_{a,i}
\]  

(66)

where \(v_i\) is the valve command, \(\theta_i\) is the joint angle, \(\dot{\theta}_i\) is the joint angular velocity, \(\tau_i\) is the joint torque, and \(v_{a,i}\) is a feedforward valve command. Subscripts \(d\) indicate desired values, and quantities denoted with \(k\) are scalar gains. This joint level servo runs at 5kHz on SARCOS and at 1kHz on ATLAS.

We augment this independent joint control by coupling joints and providing force control based on force/torque sensors in the wrists, ankles, and skin of the robot,

\[
v = K_{\theta}(\theta - \theta_d) + K_{\dot{\theta}}(\dot{\theta} - \dot{\theta}_d) + K_{\tau}(\tau - \tau_d) + K_f(f - f_d) + v_{ff}
\]

(67)

where \(v, \theta, \dot{\theta}, \tau,\) and \(v_{ff}\) are the corresponding vector quantities, and the \(K\) are matrix gains coupling all joints. \(f\) are sensed contact forces. The coupled controller runs at 1kHz on SARCOS and 500Hz on ATLAS.

Figure 1 does not include our state estimator, which provides appropriate state information to all other modules. The details of this work on state estimation are described in [http://www.cs.cmu.edu/~cga/drc](http://www.cs.cmu.edu/~cga/drc).

## 21 Operator-Exoskeleton Force Control using position controlled actuation

We will now consider position controlled actuators. At this point our dynamics formulation is no longer appropriate, since we are assuming the actuators are now position rather than torque sources.

### 21.1 Get out of the way control with position sources

The exoskeleton moves as to create as little force as possible between the operator and the exoskeleton. The actuators in this case may be position sources or force sources with high servo gains, but ultimately operator-exoskeleton forces are mapped to exoskeleton velocities or angular velocities.

\[
\dot{q}_d = J^{-1}(K_1f)
\]

(68)

where \(\dot{q}_d\) are the commanded exoskeleton joint velocities, \(J\) is an appropriate Jacobian matrix, \(K_1\) is a gain matrix, and \(f\) is the force vector to be controlled.


Inverting a Jacobian matrix is problematic when the matrix is nearly singular. Using a fixed or gain scheduled gain matrix may make more sense.

\[
\dot{q}_d = K_3f
\]

(69)
Simplifying equation 68 by assuming the forces to be controlled are directly applied to the exoskeleton joints, and thus the Jacobian matrix is the identity matrix, gives us a similar equation:

\[ \dot{q}_d = K_1 \tau \] (70)

and we avoid inverting a Jacobian and problems with singularities.

### 21.2 Operator force feedback with position sources

We introduce a desired force to allow for more complex force control.

\[ \dot{q}_d = J^{-1}(K_1(f - f_d)) \] (71)

The exoskeleton could move as to provide a scaled version of exoskeleton-world contact forces, or some other (usually simple) mapping.

\[ f_d = \alpha J_{ow} f_w \] (72)

### 22 State Estimation

We use the operator state (the set of all joint angles and velocities as well as a root position and velocity to represent operator location and orientation in the world) as the basis of inverse dynamics and online optimization-based control. In a full-body exoskeleton (not necessarily fully actuated), the exoskeleton can be used to measure operator positions, and velocities. For partial exoskeletons, it may be the case that additional sensors such as distributed angle sensors (goniometers), tachometers, or inertial measurement units (IMUs) are attached to the operator or part of the operator’s clothing. A major goal of state estimation is to process these measurements to produce a useful estimate of operator state.

We have not seen a lot of discussion of state estimation of exoskeletons in the literature. In lieu of that, we will discuss our approach to state estimation on humanoid robots here. State estimation to deal with sensor noise, disturbance estimation, and incomplete or redundant sensing is a separate process and is as independent as possible from the control method. This makes evaluating different control methods much easier. Disturbance observers for disturbances such as moving support surfaces, contact or wind forces, and model errors are also implemented.

For humanoid robot state estimation (for proposed exoskeleton state estimation) we use extended Kalman Filtering (EKF) to perform state estimation. Currently, using the full body dynamics (equations 45 or 46) and any additional constraint equations for ground contacts (feet on the ground):

\[ J_w \ddot{q} + J_w \dot{q} = 0 \] (73)

is too expensive to produce both a forward prediction and a linearized model about the current state in the desired 1ms time step. Therefore we approximate the dynamics by separating the center of mass dynamics from the joint dynamics and use two separate extended Kalman filters: a filter for the center of mass dynamics, and a filter for the joint dynamics.
We use an orthogonal (QR) decomposition to project motion into the orthogonal complement of the contact Jacobian where the “external” dynamics (the dynamics of the center of mass) are separate from the “internal” dynamics (the motion of the joints).

Joint velocities are estimated using actuator position sensors, actuator load sensors, actuator commands, a single high quality IMU at the pelvis, and multiple low quality (but cheap) MEMs IMUs spread throughout the robot or exoskeleton. We also take into account feet and hand contact forces, which are measured. We use the full internal dynamics as the process model, assuming the root dynamics are known. See Chapter 3 of [http://www.cs.cmu.edu/~xxinjile/pdf/main.pdf](http://www.cs.cmu.edu/~xxinjile/pdf/main.pdf)

The internal dynamics are linearized and covariances updated by a separate process at a 100Hz rate. The joint positions are predicted by integrating the estimated velocities.

For higher level (overall) state estimation, we developed a center of mass location and velocity estimator. The center of mass filter is modeled as a multiple model EKF with contact switching to handle changes in support: single support, double support, and off the ground. We use the robot or exoskeleton joint sensing (kinematics) to create a virtual sensor that measures center of mass position and velocity relative to a stance foot. We use Linear Inverted Pendulum Model (LIPM) dynamics with an offset to predict the center of mass motion. This offset can be interpreted as a modelling error on the center of mass position, or an external force exerted on the center of mass of the robot/exoskeleton, or a combination of both. The center of mass estimator was implemented on our Atlas humanoid robot. This estimator is especially helpful when compensating for unplanned slow changing external forces applied at unknown locations on the robot or exoskeleton, which is quite likely when operating in tight spaces. It also handles relatively small dynamic forces well when walking, e.g. dragging a tether or pushing through a spring loaded door. Thanks to the estimator, very little tuning is done for our mass model. During the DARPA Robotics Challenge Finals where no safety belay was allowed, our fall early warning system based on this estimator successfully saved our robot from falling on two occasions, and made us the only competitive team without a fall or need for a human rescue among all teams.

**Safety Code: Fall Prediction:** The most significant contribution of the external force estimator is that it can detect when a large external force is being applied that might push the robot or exoskeleton over. We compute a “corrected capture point” (CCP), which is an offset to the current capture point. The offset takes into account the estimated external force, represented as an offset to the center of mass. The corrected capture point getting close to the boundary of the polygon of support warns the controller that the robot or exoskeleton might fall if the external force is maintained. We can also compute the corrected capture point assuming that the external force follows a known time course plus an estimated constant offset, or steadily increases or decreases for a fixed time interval based on an estimated derivative. We assume the external force is due to the robot’s or exoskeleton’s movements, and not due to external disturbances such as wind, a moving support platform or external agents pushing on the robot. The current behavior is stopped and the robot or exoskeleton can safely be “frozen” in place.

A derivation of the corrected capture point starts with LIPM dynamics augmented with a true center of mass offset and a true external force:

\[
\ddot{c} = (c + c_{offset} + f_{ext} \cdot z/mg - \text{cop}) \cdot g/z = (c + \Delta - \text{cop}) \cdot g/z \quad (74)
\]
where \( c \) is the location of the center of mass projected on the ground plane, \( cop \) is the center of pressure location in that ground plane, and \( \Delta \) is the sum of the true center of mass offset from the modeled center of mass and any external horizontal force. Our extended Kalman filter estimates \( \hat{c} \), \( \hat{\dot{c}} \), and \( \hat{\Delta} \), taking into account the current center of mass height \( \hat{z} \). We assume a constant center of mass height in estimating the corrected capture point based on the estimated capture point:

\[
\hat{CCP} = \hat{CP} + \hat{\Delta} = \hat{c} + \hat{\dot{c}} \hat{z} / g + \hat{\Delta}
\] (75)

For dynamic walking, the corrected capture point goes beyond the current single foot support polygon during swing phase and is captured by the touchdown foot. One way to predict a fall is to use the swing foot time to touchdown and predict if the current capture point is within the support polygon of possible or desired footholds. This approach is complicated because it depends on the controller. Our simpler solution is to use a heuristic that detects a fall only if the corrected capture point is outside of the current support polygon for a continuous period of time. The time threshold was set to 0.6 seconds after extensive testing. As soon as a fall is predicted, there are several things the humanoid or exoskeleton can do, such as using angular momentum to maintain balance, or changing the current or future foot placements. We have implemented a simple step recovery controller that works in single support where the robot or exoskeleton corrects the next foot placement and timing using a simple heuristic to avoid self collision. We would only step left if the robot or exoskeleton is falling left, and similarly for right steps. We avoid crossing the legs in using foot placement to balance laterally.

Results of Safety Code: Robot caught on door frame: In the DRC rehearsal, the robot was caught on the door frame when sidestepping though (Figure 4). The walking controller detected an anomaly in the estimated external force in the sideways direction \( (F_x) \), delayed liftoff and remained in double support, and stopped the current behavior to allow for manual recovery (Figure 5).

Results of Safety Code: Manipulation Error: For the manipulation controller, the robot is always assumed to be in double support, and the support polygon is computed
by finding the convex hull of the foot corner points (light green in Figure 6), computed using forward kinematics. To prevent the robot from falling during manipulation, we require the corrected capture point to be within a subset of the support polygon called the safe region, (dark green in Figure 6). When the corrected capture point escapes the safe region, a freeze signal is sent to the manipulation controller, and it clears all currently executing joint trajectories and freezes the robot at the current pose, with the balance controller still running.

During our second run in the Finals, our right electric forearm mechanically failed when the cutting motion was initiated for the drill task. The uncontrolled forearm wedged the drill into the wall and pushed the robot backwards. The controller stopped the behavior (a freeze), and saved the robot from falling (Figure 6). The operator was then able to recover from an otherwise catastrophic scenario.

The time plot in Figure 6 shows candidate fall predictors during this event. We can eliminate some candidate fall predictors easily. The center of mass (CoM) (and a “corrected” CoM (not shown)) usually provide a fall warning too late, because the CoM velocity is not included. The capture point (CP) does not include information about external forces. The center of pressure (CoP) is noisy and gives too many false alarms. It can warn of foot tipping, but it is less reliable about warning about robot falling, which is not the same thing as foot tipping in a force controlled robot or if there are non-foot contacts and external forces. In this plot, we see that the CoP moves away from the safe region during recovery, predicting that the robot is falling again, while the corrected capture point (CCP) moves towards the interior of the safe region.

Unscented and particle filters could be used instead of extended Kalman filtering. It may well be the case that unscented filtering outperforms extended Kalman filtering.

22.1 Current state estimation approaches in exoskeletons

We see that Sensitivity Amplification Control uses acceleration as an input. Estimating acceleration using double differentiation with low pass filtering currently works well on real exoskeletons. However, impacts, shock waves, and the fact that neither the operator’s body parts or the exoskeletons parts are rigid bodies and the joints are not well defined suggest
this approach can be improved with actual acceleration measurements. It is now possible to distribute MEMs IMUs throughout the exoskeleton to improve both velocity and acceleration estimation which should improve Sensitivity Amplification Control.

23 Robustness to parametric modeling error and unmodeled dynamics

We need to be robust to parameter modeling errors such as the wrong mass, center of mass location, joint location, or moment of inertia for an exoskeleton part, variations in the operator parameters, and variations in attachments to the exoskeleton and payload. Examples of unmodeled dynamics (non-parametric modeling error) include: backlash, joint play, structural deformation, flexible wires and hoses, actuator dynamics, processing delay, elastic operator tissue and straps, and loose or slack operator tissue and straps.

There are two questions to focus on:

23.1 How robust are these control schemes to parametric and non-parametric modeling error?

This question needs to be addressed using simulation, and is deferred to a possible future white paper.
23.2 How can we make control more robust?

One way to understand control systems is to separate 1) what happens when there are no errors and feedback control is not active, and 2) what happens in response to errors (feedback control). What happens when there are no errors in model-based control is feedforward control, which does not have stability issues (except for some ignorable technical issues). If your model is good, you get good performance. If your model is bad, you get larger errors and have to rely on feedback control. You may crash the exoskeleton, or fall down, but you will not oscillate (except in some contrived examples) and errors do not grow exponentially in response to feedforward control errors (except for unstable and chaotic systems). So feedforward control is technically robust, and we don’t have to worry about it going wildly out of control. We do have to worry about performance. We have to worry about how to get good models, and what to do about stuff that varies: contacts, operator body, operator behavior, payloads, attachments, etc.

On the other hand, feedback control can lead to steady state or growing oscillations, which we colloquially refer to as being unstable or an instability. Since walking and running are also oscillations which we refer to as stable behaviors, the technical meaning of the word “stable” is more complex than just “it keeps moving” or “it doesn’t do what the operator wants”. In any event, we don’t want oscillations.

The best way to avoid oscillations is to manually design a conservative independent joint feedback controller. This way it can be tested independently of any feedforward control. Unfortunately, inverse dynamics and online optimization-based control mix feedforward and feedback control so we cannot use the recommended, well established, and safe approach of manual conservative independent joint feedback control design.

What to do about this to achieve robust online optimization-based control is currently a research issue. In actual implementations, skilled engineers manually tune optimization weights and the various constraints that are active to make this type of control work reliably. The fact that this has been done on many actual humanoid robots in many situations is comforting. The fact that none of these implementations have moved with the speed and agility of a top percentile human athlete is disconcerting.

We can “verify” designs using simulation, but making accurate simulations of the effects of contact and unmodeled dynamics is also a current research issue. There is no getting around the fact that our operators will be like the test pilots of early planes. The exoskeletons may crash, flail wildly, or oscillate. We need to plan for operator safety from day 1 of operation, not try to retrofit it later.

If the robustness of online optimization is not adequate, an alternative is combining feedforward and feedback control into one function and using gradient descent in simulation to improve the control robustness. One technique to do this is multiple-model design. http://www.cs.cmu.edu/~cga/papers/acc2012-tr.pdf

Note that adaptive or learning control does not promise to improve the kind of robustness exoskeletons need. These approaches promise to improve performance, and try to track (slow) changes in the operator, exoskeleton, or contacts to maintain performance. If something changes quickly, we need robust control design.
24 Specific Proposed and Implemented Exoskeleton Control Schemes

24.1 Sensitivity Amplification Control

Sensitivity Amplification Control (SAC) used to control BLEEX is a dynamic cancellation technique (a.k.a. inverse dynamics, computed torque, or feedback linearization) that also tries to modify the apparent inertia of the exoskeleton (equation 17) using acceleration feedback. It is not clearly stated but it appears that the acceleration is the result of double differentiating position, as there do not appear to be velocity sensors on BLEEX. Low pass filtering is applied to reduce the high frequency noise amplified by the double differentiation process.

24.2 Integral admittance control

Integral admittance control is a refinement of sensitivity amplification control. This section quotes from:

![http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=7139989

“[Exoskeletons] have the implicit property of causing a virtual modification of the dynamic response of the human limb. We use this property of the exoskeletons action to formulate a unified control design framework called Integral Admittance [torque to angle] Shaping, which designs exoskeleton controllers capable of producing the desired dynamic response for the assisted limb. In this framework, a virtual increase in the admittance of the limb is produced by coupling it to an exoskeleton that exhibits active behavior. Specifically, our framework shapes the magnitude profile of the integral admittance (i.e. torque-to-angle relationship) of the coupled human-exoskeleton system, such that the desired assistance is achieved. This framework also ensures that the coupled stability and passivity are guaranteed.

“... the impedance of the coupled human-exoskeleton system needs to be reduced below that of the unassisted human limb. This implies that the exoskeleton needs to cancel its own impedance first and then compensate for at least a part of the human limbs impedance. Therefore, the desired exoskeleton behavior must be that of a negative impedance ... Consequently, the feedback gains will all be positive. In other words, the exoskeleton controller uses positive feedback, and hence the exoskeleton exhibits active behavior, which is capable of performing net positive work on the limb. ... However, positive feedback naturally raises the question of stability, and so we now explain how coupled stability can be achieved. Although the exoskeleton exhibits active behavior, which can be potentially destabilizing, the controller can be designed such that the coupled human-exoskeleton system is stable and passive. ...

“... Inertia compensation is more complex ... It can be shown that using only positive acceleration feedback ..., the gain margin of the coupled system reduces to the moment of inertia of the exoskeleton, which implies that the exoskeleton controller ... can at the most compensate for the exoskeletons own moment of inertia before going unstable. This implies that the moment of inertia of the coupled human-exoskeleton system cannot be reduced below that of the unassisted human limb, without compromising coupled stability. However, using low-pass filtered acceleration feedback, it can be shown that inertia reduction can be achieved.
This work uses filtered acceleration feedback with a second-order low-pass Butterworth filter. This is a form of frequency domain virtual model control.

Kalman filter:

24.3 Dual Control Approach

This approach implements a passive leg swing in addition to Sensitivity Amplification Control for the stance leg.

“The robot utilized the dual-mode control scheme, which is comprised of the active control for the stance phase and the passive control (using bypass valves) for the swing phase, to achieve high walking speed in the swing phase while supporting heavy loads in the stance phase. To reduce the sudden change of the torque command at the transition from the swing phase to the stance phase, a smoothing method is adopted. We also implemented a pre-transition method to take a foot off quickly for fast walking by predicting the change from the swing to the stance in advance.”

Stance: virtual joint torque control method Kazerooni et al.(2005) When contact location between the wearer and the exoskeleton is not fixed and difficult to estimate, this method has been shown to be an effective method to generate the locomotion for an exoskeleton robot.


To reduce sudden changes(command jump) at the phase transitions, a smoothing method is introduced and a pre-transition method is used to solve the swing delay due to the internal pressure(approx 5 bar).

oonbum Bae, Kyoungchul Kong, Masayoshi Tomizuka Gait Phase-Based Control for a Rotary Series Elastic Actuator Assisting the knee

Transition Control

1) Smoothing Method: During transitions of the gait phase, discontinuity of the control command torque are occurred by the different condition for the fixed coordinate which is on the backpack at the swing phase or the foot at the stance phase. In this paper, to reduce this sudden change due to gait phase changes, a smoothing method is proposed as shown in Fig. 7. An exponential function is considered as the weighting function for smoothing as shown in (6). In this case, the weighting is small at the initial stage but it exponentially converges to one for supporting the load quickly.

2) Pre-transition Method: The pre-transition method is that the passive mode is executed in the pre-swing phase prior to toe off. This dramatically reduces the moving
24.4 Ground Reaction Force Control

The ground reaction forces (GRF) magnitude and direction is used to command the actuators. In some research the GRF sensors are used together with other sensors in the control architectures [17], while in other exoskeleton the control system is based on the GRF merely [RoboKnee, Honda].


24.5 Virtual model control

The goal here is to make the exoskeleton imitate a desired (usually nonlinear) dynamic system, For running it could be a pogo stick or trampoline, for example.

There are several possible versions of this type of control:

1. Map from operator-exoskeleton contact forces and system state to exoskeleton actuator or internal forces.
2. Map from operator-exoskeleton contact forces and system state to exoskeleton acceleration.
3. Map from operator-exoskeleton contact forces and system state to exoskeleton-world contact forces.
4. Map from exoskeleton-world contact forces and system state to exoskeleton actuator or internal forces.
5. Map from exoskeleton-world contact forces and system state to exoskeleton acceleration.
6. Some combination of the above.

virtual model approach: hopper, compass gait (J. Pratt)
estimate trajectory (jumping) vs. program behavior (trampoline, hopper, compass gait)
- J’ control + behavior
- Inverse dynamics + behavior

24.6 Task Specific Control

Task specific control can involve switching low level control modes, or abstracting the behavior of the exoskeleton with a hierarchy:

1. Task specific control: Generate desired motions and contact forces between the operator and the exoskeleton, and the exoskeleton and the world.
2. Sometimes there are intermediate levels of control.
3. Exoskeleton control: Control the exoskeleton to generate the desired motions and desired contact forces, with the desired impedance or admittance.

25 Conclusions and Recommendations

0) We recommend a red team that explores traditional frequency domain control system design, as well as an emphasis on advanced control approaches.

1) We recommend a particular overall control architecture in Section 6. We do not expect this to be controversial.

2) We recommend actuation that can be treated as force or torque sources. This is a potential problem for revolute electric actuation, as it is difficult to get necessary performance at low gear ratios (20-50) and fit the necessary torque sensing. We do not expect this to be controversial.

3) We recommend the use of dynamic models of the exoskeleton, as has been done for the last decade starting with the BLEEX exoskeleton. We do not expect this to be controversial.

4) We recommend using online optimization as part of the Low Level Controller in the form of quadratic programming. Online optimization was tested by the top biped robot teams in the DARPA Robotics Challenge, and worked well. A possible objection to online optimization is that it requires more substantial computing resources than current exoskeleton control methods. We feel the feasibility of this approach has been demonstrated and is easily achievable. Perceptual computing costs will dwarf computing costs for control, so the computation cost is a non-issue.

5) We recommend applying as many sensors as possible, and asking the performers to make it easy to add more by making the sensor network available in the design. This maximizes the probability of success with respect to control architectures and algorithms by enabling multiple control approaches to be implemented, refined, and support each other. In particular, we would like to see force sensing between the operator and the exoskeleton, including at the feet, as much force sensing between the exoskeleton and the world as possible, but minimally full six-axis force/torque sensing at the exoskeleton feet (where they touch the ground). We would like to see multiple MEMs IMUs (measuring linear acceleration and angular velocity) installed across the exoskeleton and at least one high quality fiber optic gyro IMU. We would like to see high quality direct velocity sensing such as high count (100,000 counts/revolution) encoders and/or analog rotary or linear tachometers on actuators and joints. We would like to see actuator force or torque sensors (load cells or equivalents) with the measurement on the link side (rather than the actuator side) of any transmission. Electric current in motors and oil pressure in hydraulic pistons can also be used for actuator force estimation, but because these measurements are on the actuator side before the transmission and in the case of hydraulics before the oil seals on the piston, these measurements are greatly contaminated by friction. On the Atlas humanoid we typically saw 10Nm joint torque estimation errors for a system that estimated actuator output using oil pressure on each side of the piston head. A possible objection to this is additional cost. We feel it is a false economy to skimp on sensing. Leaving practical sensing out greatly increases the risk of poor performance.

6) We recommend a “symbiotic” control system design should be used as a backup, in
case more aggressive design philosophies such as “invisible” and “natural” control system designs are not achievable. The “symbiotic” control system design expects the operator to adapt to the exoskeleton and its control, and has the control customized for the operator.

7) We recommend making use of virtual model control specialized for the various tasks. This can greatly increase operator/exoskeleton system performance over an exoskeleton that just carries a load and is otherwise “invisible”.