Operating High-DoF Articulated Robots Using Virtual Links and Joints

Marsette A. Vona

Northeastern University College of Computer and Information Science, Boston MA
vona@ccs.neu.edu

Summary. This chapter presents the theory, implementation, and application of a novel operations system for articulated robots with large numbers (10s to 100s) of degrees-of-freedom (DoF), based on *virtual articulations* and *kinematic abstractions*. Such robots are attractive in some applications, including space exploration, due to their application flexibility. But operating them can be challenging: they are capable of many different kinds of motion, but often this requires coordination of many joints. Prior methods exist for specifying motions at both low and high-levels of detail; the new methods fill a gap in the middle by allowing the operator to be as detailed as desired. The presentation is fully general and can be directly applied across a broad class of 3D articulated robots.

1 Introduction

Normally the connectivity structure of an articulated robot's links and joints—its kinematic topology—is considered invariant.¹ This chapter explores a new kind of human interface based on virtually making such changes, by allowing virtual links and joints to be interactively connected to a virtual model of the robot. We call such virtual kinematic constructions *virtual articulations*.

We present the architecture of an implemented simulator [38] for arbitrary topology 3D articulated robots with virtual articulations, and show how it can be applied as an intuitive human interface to operate coordinated motion in robots with many joints. Experimental results are presented with the 36 degree of freedom (DoF) All Terrain Hex Limbed Extra-Terrestrial Explorer (ATHLETE) [41] and with a simulated modular mobile tower structure with 100s of joints [8].

Due to their application flexibility, high-DoF robots are increasingly common: humanoids with 20 or more DoF are now available off-the-shelf, snake-like robots with 10s of joints have long been studied, and modular/self-reconfiguring robots can have 10s to 100s of actuated joints. This flexibility is especially attractive in space exploration, where the extreme costs of transportation from Earth are balanced by maximizing versatility in the delivered hardware. But remote operation of complex

¹ Robots capable of on-line reconfiguration exist, but are unusual.

robots can be tedious; a main feature of this chapter is a set of examples where virtual articulations are used to operate inspection and manipulation tasks on the ATHLETE hardware that would have been challenging in prior interfaces. We also show an application to a simulated modular mobile tower that could be useful in space, where structures need to be assembled from compact storage. If such structures were controllably mobile and deformable they could simultaneously support manipulation, inspection, and locomotion, thus greatly reducing costs relative to launching separate robotic hardware.

Though some of the results in this chapter have been reported previously [8, 9, 36, 37], here we include for the first time the details of the multi-priority system architecture which enables motion constraint with virtual articulations and scaling to 100s of joints.

Section 2 explains the idea of virtual articulations with some examples. We then review related work in Section 3, and present the system architecture in Section 4. Section 5 gives several detailed application examples, followed by conclusions in Section 6.

2 Virtual Articulations

What makes operating high-DoF robots difficult? One issue is that there are often many ways the robot could move to achieve a task (kinematic redundancy) and some motions may be preferred due to secondary goals [28]. Sometimes a human operator can quickly visualize a desired motion, but expressing it in an operations system can be tedious. The approach described in this chapter allows the operator to graphically and intuitively specify motions using virtual articulations. Task-specific coding and low-level motion sequencing is minimized. It can also be time consuming to specify motion for large systems with many simultaneously moving DoF, even when motions for subsystems are easily defined, simply due to scale. In a technique we call *structure abstraction*, virtual articulations can be applied to impose hierarchical organization, and can enable re-use of substructure motion constraints.

To see how virtual articulations can help, consider an example with ATHLETE. With 36 revolute joints in six identical limbs, ATHLETE can be applied in a broad range of tasks. Often, an intended motion is easily visualized, but expressing it could be tedious—even using custom-built direct manipulation hardware [22]. Figure 1 shows one example of how virtual articulations can help: the operator would like to tilt the field of view of an un-actuated camera by rotating the robot’s body about a virtual axis in space, with the legs moving as necessary. Using virtual articulations, the intended rotation axis can be defined by attaching a virtual revolute joint between the robot body and the ground. Rotating the joint drives the intended motion, and the system automatically computes compatible motions for all the legs.

Virtual articulations can also help organize large systems with 10s to 100s of DoF. Abstraction is well known for managing complexity in algorithms, but so far there have been few corresponding techniques in kinematics. By applying virtual articulations in a specific hierarchical framework, *structure abstraction* can be used to effectively hide the complexity of a kinematic subsystem in a way that corresponds to traditional algorithmic abstraction. Figure 2 shows a simple example where a series chain of four revolute joints is abstracted as a piston-like assembly, hiding the motion of the middle link. This assembly is later used as a subsystem in Section 5.3.

3 Research Context

A few related ideas have previously been reported. For example, Flückiger [14] shows a virtual interface where a model of the robot is provided to the operator for pose manipulation. Our approach fundamentally differs from this type of system in that we allow the operator to virtually change and augment the kinematic structure, in addition to moving it.

In [26, 27], Pratt et al introduced *virtual model control*, where virtual physical dynamic elements—often springs and dampers, but in a few cases also kinematic joints—are added to a robot model as a means to design motion control strategies. In their reports, the virtual elements are manually designed, tuned, and implemented on a case-by-case basis for each task of each mechanism. Overall, physics based approaches require more parameter tuning than the purely kinematic models we use, potentially adding additional burden on the operator as virtual elements are added/removed/reconfigured. And, while algorithms for efficiently simulating the physical dynamics of reduced coordinate models and closed kinematic chains [12] are known in the literature, currently available real-time physics engines, such as ODE [33], are mostly based on Cartesian coordinate models (called “maximal coordinate methods” in [20]) and need to be carefully tuned to efficiently and stably simulate kinematic chains, especially in the presence of chain-closing joints.

Geometric constraints have been proposed for defining and constraining motion in figure animation [2, 25, 40]. In these works typically only a few simple constraints are available, for example, point-on-plane and point-on-line. The constraints are not themselves considered to be kinematic joints, and can only be added between geometric objects situated within links of the original articulated model. In our approach the operator is free to construct arbitrary additions including both virtual links and a variety of virtual joint types.

Compared to other operator interface approaches, virtual articulations fill a gap between existing low-level methods to operate articulated robots, including forward and inverse kinematic control, and existing high-level methods such as goal-based motion planning and programming-by-demonstration.

Arguably the simplest low-level methods are traditional forward- (joint space) or inverse-kinematic (task space) operation. Without higher-level goals or constraints, both are potentially tedious in the high-DoF case. Task priority and task space augmentation approaches [5] can maintain such constraints, but do not themselves offer a convenient way to *design* them.

A few authors have previously explored the idea of constraining the kinematic motion of redundant robots by adding chain-closing virtual joints in specific cases. Ivlev and Gräser [18, 19] defined virtual chain closures to restrict the motion of redundant revolute serial-chains to avoid known obstacles in the environment. They treated only smaller examples which could be solved analytically on paper. Bruyninx [4] also showed an example where a virtual closed chain minimizes the otherwise unconstrained twist motion about the axis of a round tool on a revolute-jointed industrial manipulator.

Another related idea is to introduce optimization metrics, such as energy minimization, joint limit avoidance, or manipulability maximization [21, 28]. These are useful but blunt instruments—there is no direct way for the operator to arbitrarily customize the intended motion. Virtual articulations are one language for such customization: the operator can constrain motion as much or as little as desired, and

a large class of motions can be developed starting from a relatively small catalog of virtual joints. Traditional optimization metrics can still be added as additional priority levels alongside virtual articulations.

Turning to high-level methods, goal-based motion planning, the “piano moving” problem of achieving a target configuration among obstacles, is typically not directly applicable when the operator would also like to specify detailed aspects of the motion. Again, something more is needed if the operator needs to arbitrarily customize the motion *on the way* to a goal configuration.

Another high-level technique, programming-by-demonstration, allows more specific motion specification, but is hard to apply when the robot topology diverges from preexisting systems (including known animals). Thus it has been used with some success for humanoids (e.g. [24]), or when mimicking hardware is available, as in some of our earlier work with ATHLETE [22]. Virtual articulations are not tied to any particular topology. And the experiment in Figure 12 shows that that an integration of programming-by-demonstration with virtual articulations can have some of the advantages of both methods.

There seem to be only a few prior works related to our concept of structure abstraction. Davis [7] explored spatial geometric abstraction at a conceptual level, including an interesting but brief mention of a “kinematic device as black box.” Zanganeh and Angeles [44] give a more detailed and formal development of the idea of separating topologically large kinematic structures into sub-mechanisms, but their approach does not specifically separate interface from implementation—it just demarcates sub-mechanisms. Our structure abstraction includes this capability, but also allows interface mechanisms that stand-in for underlying implementations.

4 System Architecture

Articulated systems, which we call *linkages*², are composed of links (rigid bodies) interconnected by joints. The binary connectivity of joints and the arbitrary connectivity of links naturally suggest representing the topology of a linkage as a graph, as in Figure 3, where the links are vertices and the joints are edges. Such graphs have been used at least since the 1960s [11].

The 3D kinematic simulation system we present here—the *Mixed Real/Virtual Simulator and Interface* (MSim)—allows the construction of arbitrary topology linkages. Unlike some prior kinematic simulators [3, 10, 29, 43], to support interactive construction of virtual articulations we must support both on-line structure changes (adding, removing, reconnecting joints) and closed kinematic chains (cycles in the linkage graph). Figure 4 shows the architecture of our prototype implementation, which we have made freely available in source and executable form [38].

When the linkage graph is cyclic there may be multiple paths between a given pair of links; our system always maintains a spanning tree of the graph, so that the relative pose of any two links can always be uniquely defined by the path between them consisting only of *tree joints*; the remaining joints, which complete kinematic cycles, are called *closure joints* or just *closures*. This tree also induces a natural di-

² Some authors use this term more specifically, e.g. only for 1-DoF systems, or for systems with only revolute joints.

rectivity on the edges it contains: such edges point towards the root of the tree. (The same edge direction also gives the forward sense of the associated joint transform.)

How do linkage graphs arise and evolve? Algorithms for topological mutation of kinematic graphs have not often been presented in detail, in part because robots do not often change topology. But we must add, remove, and reconfigure virtual articulations on-line. We have thus developed a set of *structure mutation* primitives (table 2) for common kinds of structural (topological) changes. These are sufficient to build any linkage structure from scratch. We omit the algorithms here due to space constraints, see [39] for details.

The following formal definition of the components of a linkage and their properties will be helpful (also see Figure 3).

Definition 1. *Each joint (graph edge) j is directed to connect links p_j and c_j the parent and child link of j . The edge direction is from c_j to p_j .*

Definition 2. *Each link k (graph vertex) has an associated coordinate frame F_k called the link frame of k . The link frame of the child (resp. parent) link of a joint j is called the joint's child (resp. parent) frame F_{c_j} (resp. F_{p_j}).*

Definition 3. *For each joint j there is an associated rigid-body joint transform X_j that defines the pose of the child frame relative to the parent frame.*

Definition 4. *The joint transform X_j must lie within the joint mobility space $\mathcal{M}_j$, a subspace of the full 6-dimensional space of 3D rigid body transforms (the special Euclidean group $SE(3)$) depending on the joint's type and limits.*

For example, a revolute joint allows only motion within a 1-dimensional subspace of $SE(3)$ —rotation about a fixed axis. Limits, if specified, further restrict motion to a closed bounded interval of this subspace.

Our joint mobility representation, which is based on the exponential map $(\mathbf{t}, \boldsymbol{\theta}) \in \mathbb{R}^6$, is fully described in [39]. Here we present only the final catalog of 11 joint types available in the system (Figure 5). These include all lower-pair joints except helical; i.e. revolute, prismatic, cylindrical, spherical, and planar. In addition, some higher pairs are included for convenience: a *pin-slider* joint combines a prismatic and revolute joint with orthogonal axes; a *point-slider* is similar but replaces the revolute joint with a spherical joint; the *point-plane* joint combines a planar and a spherical joint; *Cartesian-2,3* joints are serial chains of two or three orthogonal prismatic joints; and finally a *general* joint allows full 6-DoF motion (and is equivalent to a serial chain of three orthogonal prismatic joints followed by a spherical joint).

Definition 5. *The degrees of freedom (DoF) f_j of a joint is the dimension of the joint space: $f_j = \dim(\mathcal{M}_j)$. The degrees of invariance (DoI) is $i_j = 6 - \dim(\mathcal{M}_j)$. The state of a tree joint t is a vector $\mathbf{x}_t \in \mathbb{R}^{f_t}$; the state of a closure joint c is always taken as its full transform $\mathbf{y}_c \in \mathbb{R}^6$.*

We define joint types as axis-aligned subspaces of the exponential map parameterization of $SE(3)$, so a tree joint state $\mathbf{x}_t$ is always just a subset of the full 6-DoF $(\mathbf{t}, \boldsymbol{\theta})$ vector. (Constant structural transforms in the linkage still allow the joint mobility directions to be posed arbitrarily.)

Definition 6. *Exactly one of the links is identified as the ground link g .*

Definition 7. Every link $k \neq g$ has exactly one of its outgoing joints identified as its parent joint p_k , and g has no parent joint. The incoming joints of a link k are called child joints of k .

Definition 8. The set T of all parent joints of all links forms a directed spanning tree with root g ; all edges in T point towards g . A transform $X_{k_0 \leftarrow k_1}$ gives the pose of any link frame F_{k_1} relative to any other link frame F_{k_0} by following the unique path from k_0 to k_1 in T :

$$X_{k_0 \leftarrow k_1} = \left(\prod_{i \in g \leftarrow k_0} X_i \right)^{-1} \left(\prod_{i \in g \leftarrow k_1} X_i \right). \quad (1)$$

Definition 9. The support S_j of a joint j is the minimal ordered sequence of tree joints from the parent link p_j to the child link c_j . S_j is partitioned into the downchain $S\downarrow_j$ of joints from p_j to the least common ancestor link³ $\text{LCA}(p_j, c_j)$ and the upchain $S\uparrow_j$ from $\text{LCA}(p_j, c_j)$ to c_j :

$$S_j = (S\downarrow_j, S\uparrow_j). \quad (2)$$

Definition 10. If J is the set of all joints in a linkage with tree joints $T \subset J$, then the remaining joints $C = J \setminus T$ are chain closures.

The mobility transform X_j for a closure joint j with support $S_j = (S\downarrow_j, S\uparrow_j)$ is computed from the mobility transforms of the supporting tree joints based on Eq. 1 (here we omit the constant joint posing transforms for brevity):

$$M_j = \left[\left(\prod_{i \in S\uparrow_j} X_i^{-1} \right) \left(\prod_{i \in S\downarrow_j} X_i \right) \right]^{-1} \quad (3)$$

with the products taken in chain order.

Let L be a linkage with joints J and tree joints $T \subseteq J$ with ordering

$$T = (t_0, \dots, t_{|T|-1}). \quad (4)$$

Definition 11. The mobility space $\mathcal{M}$ of a linkage L is the product space of the individual tree joint mobility spaces, in order: $\mathcal{M} = \prod_{0 \leq i < |T|} \mathcal{M}_{t_i}$.

Definition 12. The number of degrees of freedom DoF f of L is the dimension of $\mathcal{M}$: $f = \dim(\mathcal{M})$. The number of degrees of invariance DoI i of L is the sum of the DoI of the closure joints in L : $i = \sum_{j \in J \setminus T} (6 - \dim(\mathcal{M}_j))$.

Definition 13. The tree state $\mathbf{x}$ of L is the concatenation of the states of its tree joints, in the order given above

$$\mathbf{x}^T = (\mathbf{x}_{t_0}^T, \dots, \mathbf{x}_{t_{|T|-1}}^T) \in \mathcal{M} \subset \mathbb{R}^f. \quad (5)$$

³ I.e. the topologically nearest link that is an ancestor of both p_j and c_j .

Similarly, let

$$C = (o_0, \dots, o_{|J \setminus T|-1}) \quad (6)$$

be an ordering of the closure joints $J \setminus T$ of L .

Definition 14. The closure state $\mathbf{y}$ of L is the concatenation of the states $\mathbf{y}_{o_j}$ of its closure joints in order, and the invariant error $\mathbf{e}_i$ of L is the concatenation of the invariant errors of the closure joints in order

$$\mathbf{y}^T = (\mathbf{y}_{o_0}^T, \dots, \mathbf{y}_{o_{|C|-1}}^T) \in \mathbb{R}^{6|C|} \quad (7)$$

$$\mathbf{e}_i^T = (\mathbf{e}_{i_{o_0}}^T, \dots, \mathbf{e}_{i_{o_{|C|-1}}}^T) \in \mathbb{R}^i. \quad (8)$$

$\mathbf{e}_i$ is a projection Π_i of $\mathbf{y}$,

$$\mathbf{e}_i = \Pi_i \mathbf{y}, \quad (9)$$

with Π_i an $[i \times 6|C|]$ binary matrix mapping $\mathbf{y}$ to $\mathbf{e}_i$ by selecting only the DoI of each closure joint.

Definition 15. The forward kinematic mapping (Figure 6) FK of L computes $\mathbf{y}$ given $\mathbf{x}$

$$\text{FK} : \mathcal{M} \rightarrow \mathbb{R}^{6|C|} \quad \text{FK}(\mathbf{x}) = \mathbf{y}, \quad (10)$$

and is computable by composing appropriate instantiations of Eq. 3.

The differentiability of the underlying operations implies that FK is itself differentiable; an algorithm to compute derivatives of FK is given in [39].

4.1 Hierarchical Linkages

Many real-world robots contain repeated kinematic sub-structure, e.g. multiple copies of identical legs. When possible, breaking a model up into decoupled components can speed up motion computations (Section 4.1). Further, subdivision is important in using virtual articulations in an operator interface for high-DoF mechanisms—in many cases the operator can specify motions independently for smaller parts of the model, and thus break up the job of motion specification. Taking this a step further, virtual articulations combined with model hierarchy enable our technique of *structure abstraction*, where a complex kinematic subsystem is abstracted as a simpler virtual mechanism capturing the intended motion.

The mixed real/virtual interface supports these kinds of subdivisions by structuring the overall model into a hierarchy (tree) of properly nested *sub-linkages*. Figure 7 shows an example; details are given in [39] and [37]. We allow sub-linkages to be *driving* (considered rigid with respect to their parent), *driven* (parent considered rigid), or *simultaneous* (grouping only). *Crossing joints* span the boundary between a parent and child sub-linkage.

Sub-linkages and virtual articulations can be used to implement a version of abstraction in the domain of kinematics.

Definition 16. Structure abstraction is achieved by (a) encapsulating a connected part A of a linkage L s.t. A becomes demarcated as a simultaneous sub-linkage of L , and (b) substituting a virtual linkage I for A in L , with I simultaneous in L and A a driven sub-linkage of I . A is the abstracted sub-linkage (the “implementation”) and I is its interface linkage.

Figure 7 shows an example. The concept parallels traditional abstraction in computing: I should be simpler than A , but capture all of the behavior of A that would be relevant to the surrounding mechanism.

Making A a driven sub-linkage of I is not the only option. For example, keeping A simultaneous could also make sense. However, our choice to make A driven defines a strong form of abstraction: motion of L , with I substituting for A , is *independent* of the actual motion of A . This permits A to be effectively decoupled, a simplification that can be helpful both (a) to an operator designing a motion for a high-DoF robot, and (b) to the system as it computes motions for a large linkage.

This power has a trade-off: there is no built-in constraint to ensure that A can reach every configuration to which it may be driven by I .

Definition 17. I is a proper abstraction of A if all of the reachable configurations of I drive reachable configurations of A .

It would be desirable to have an efficient algorithm that could determine, for any L , I , and A , whether I is a proper abstraction. This may be possible in some special cases, but unfortunately, the general case is easily shown to be hard; the details are given in [37].

Thus, if a proper abstraction is required, it is up to the operator to ensure it, for example by setting sufficient joint limits in I . However, even improper abstractions can be useful when combined with the constraint prioritization features of our system (Section 4.3)—if the closure invariants in A are given higher priority than those of the crossing joints between I and A , then the motion of A will remain feasible. It will not exactly match the driving links in I , but because they are virtual, this can be acceptable.

4.2 Local Assembly and Differential Control

We now turn to the simulation of linkage motions, which we achieve by a task-priority solver we call prioritized damped least squares (PDLS), along with a specific design of five priority levels.

We define two motion problems. *Local assembly* needs to be solved after the linkage structure is changed (Figure 8)—the prior linkage state will not generally be compatible with the new structure, and needs to be moved to satisfy the new constraints. The *differential control* (adapted from [16]) problem generalizes both forward and inverse-kinematic control, and supports various modes of operating mixed real/virtual structures including click-and-drag interaction (Figure 9) and trajectory following.

The PDLS algorithm uses the differentiability of the exponential map representation of joint pose to make local linear approximations of the forward kinematic mapping, and then iteratively solves local assembly, differential control, and kinostatic simulation by gradient descent in a least-squares sense with an SVD-based pseudoinverse. Nullspace projection is used for prioritizing the various constraints, which may conflict.

PDLS derives from the task-priority approach first proposed by Siciliano and Slotine in [32], with more recent improvements from Baerlocher and Boulic in [1]. We do not repeat these results here, see [1, 39] for details. Instead we focus on our contribution: a set of five specific priority levels that together solve the local

assembly and differential control problems (in [39] we also insert a sixth level that simulates quasistatic potential energy minimization).

Because the time complexity of PDLS is typically quadratic in the number of joints [39], it is important to break up large linkages into independently solvable components when possible. These are then solved independently in ordered *rounds*, which arise due to the driving/driven relationships of sublinkages. We have developed an algorithm that analyzes the topology and structure of a linkage and attempts to break it into smaller parts which can be handled separately; details are given in [37, 39].

4.3 The Five Priority Levels

The inputs to PDLS are, for each priority level l , a *level height* h_l , a *residual vector* $\mathbf{e}_l \in \mathbb{R}^{h_l}$, and an $h_l \times f$ level Jacobian matrix J_l , which relates a small change in the tree state $\mathbf{x} \in \mathbb{R}^f$ (f is the total number of DoF of the linkage) to a change in the level residual. We next derive the residuals and Jacobians for each of our five proposed priority levels: invariant, limit, lock, target, and posture. Table 1 summarizes them.

The highest priority *invariant* level solves the local assembly problem by minimizing the *invariant residual* $\mathbf{e}_i$. The forward kinematic mapping FK (Eq. 10) and the invariant error projection matrix Π_i multiply to give the forward invariant error mapping FK_i (Eq. 9) taking the current tree state $\mathbf{x}$ to the resulting invariant error $\mathbf{e}_i$:

$$\text{FK}_i : \mathbb{R}^f \rightarrow \mathbb{R}^i \quad \mathbf{e}_i = \text{FK}_i(\mathbf{x}) = \Pi_i \text{FK}(\mathbf{x}). \quad (11)$$

FK_i' is locally modeled in the neighborhood of $\mathbf{x}$ by its Jacobian matrix

$$J'_i(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} \text{FK}_i'(\mathbf{x}). \quad (12)$$

Writing $J'_i(\mathbf{x})$ stresses the dependence of the Jacobian matrix both on FK_i' and on $\mathbf{x}$, but for brevity we usually write simply J'_i , with the latter dependency implied. J'_i can be computed by multiplying the Jacobian J'_y of the full forward kinematic mapping $\text{FK}'(\mathbf{x})$ by the projection matrix Π_i :

$$\text{with } J'_y = \frac{\partial}{\partial \mathbf{x}} \text{FK}'(\mathbf{x}) \quad (13)$$

$$J'_i = \frac{\partial}{\partial \mathbf{x}} \text{FK}_i'(\mathbf{x}) = \Pi_i \frac{\partial}{\partial \mathbf{x}} \text{FK}'(\mathbf{x}) = \Pi_i J'_y. \quad (14)$$

We will cover the limit and lock levels below; for now, skip to the *target* level, which solves the differential control problem. Consider the motion of a linkage after a user has moved a link (Figure 9). If the link is not part of any closed chains, then the forward kinematic mapping will not be affected. Otherwise, to keep $\mathbf{e}_i$ (breakage of closure joints) minimized, the system may be required to not (fully) move the link as the user has requested, and/or it may be necessary to move some joints to compensate.

Let the total number of DoF of the closure joints in a linkage L with closure joints C be $d = 6|C| - i$, and define a new $[d \times 6|C|]$ binary projection matrix Π_{fc} which compliments Π_i by selecting only the DoF from $\mathbf{y}$. Using this, form a vector $\mathbf{z}$ comprising all the DoF of both the tree and the closure joints in a linkage L :

$$\mathbf{z}^T = \left(\mathbf{x}^T, (\Pi_{fc}\mathbf{y})^T \right) \in \mathbb{R}^{f+d=z}. \quad (15)$$

In general, the operator is not restricted to move any single joint, but may manipulate any subset of joint DoF. Further, they may also operate *link DoF*, which are defined simply as the DoF of a general (6-DoF) virtual closure joint automatically attached between the link and the ground, neatly handling inverse kinematic control. Let t be the total number of DoF under control, and let $\mathbf{t} \in \mathbb{R}^t$ be the current *target waypoint*. To follow a path, $\mathbf{t}$ can be periodically replaced by the operator—e.g. by dragging the mouse—with the system either interpolating intermediate waypoints or just switching directly from one to the next.

Let Π_t be a $[t \times z]$ binary projection matrix selecting the t current target DoF from the z total DoF. The *target residual* is

$$\mathbf{e}_t = \Pi_t \mathbf{z} - \mathbf{t} \quad (16)$$

and the *forward target error mapping* FK_t is

$$\text{FK}_t : \mathbb{R}^f \rightarrow \mathbb{R}^t \quad \mathbf{e}_t = \text{FK}_t(\mathbf{x}) = \Pi_t \begin{bmatrix} \mathbf{x} \\ \Pi_{fc}\text{FK}(\mathbf{x}) \end{bmatrix} - \mathbf{t}. \quad (17)$$

The *target Jacobian* is the $[t \times f]$ matrix of partial derivatives of 17:

$$J_t = \frac{\partial \mathbf{e}_t}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} \text{FK}_t(\mathbf{x}) = \Pi_t \begin{bmatrix} \mathbf{I}_f \\ \Pi_{fc} J_y \end{bmatrix} \quad (18)$$

with $\mathbf{I}_f$ the $[f \times f]$ identity matrix.

The lowest-priority *posture* level derives from the posture variation technique described in [1], drawing the joints toward a lowest-priority default pose. However, [1] only covered tree joint DoF; our formulation, which handles closure as well as tree joints, parallels target tracking. The *posture waypoint* is a vector $\mathbf{p} \in \mathbb{R}^p$ and Π_p is a new $[p \times z]$ binary projection matrix s.t.

$$\mathbf{e}_p = \Pi_p \mathbf{z} - \mathbf{p}. \quad (19)$$

However, in this case the Jacobian will not be a $[p \times f]$ matrix, because the posture variation components corresponding to tree joint DoF will be handled directly by the final nullspace projection [1]. $\mathbf{e}_p$ must be partitioned into two sub-vectors, one for closure joint DoF and one for tree joint DoF, which will be handled separately:

$$\mathbf{e}_p^T = \left(\mathbf{e}_{pt}^T, \mathbf{e}_{pc}^T \right) \quad (20)$$

s.t. p_t, p_c are the lengths of $\mathbf{e}_{pt}$ and $\mathbf{e}_{pc}$, respectively, with $p = p_t + p_c$. Then the posture variation vector is

$$\Delta \mathbf{x}_p = -\mathbf{e}_{pt}. \quad (21)$$

$\mathbf{e}_{pc}$ is the *posture residual* vector and J_{pc} the *posture Jacobian*, computed by the *forward posture error mapping* FK_{pc}

$$\text{FK}_{pc} : \mathbb{R}^f \rightarrow \mathbb{R}^{p_c} \quad \mathbf{e}_{pc} = \text{FK}_{pc}(\mathbf{x}) = \Pi_{pc} \Pi_{fc} \text{FK}(\mathbf{x}) - \mathbf{p}_c \quad (22)$$

$$J_{pc} = \frac{\partial \mathbf{e}_{pc}}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} \text{FK}_{pc}(\mathbf{x}) = \Pi_{pc} \Pi_{fc} J_y. \quad (23)$$

Π_{pc} and $\mathbf{p}_c$, used in the above equations, are taken from corresponding tree/closure partitions of Π_p and $\mathbf{p}$:

$$\Pi_p = \begin{bmatrix} \Pi_{pt} & \\ & \Pi_{pc} \end{bmatrix} \quad (24)$$

$$\mathbf{p}^T = (\mathbf{p}_t^T, \mathbf{p}_c^T). \quad (25)$$

We now return to the *limit* level. As for posture variation, [1] also presented a technique for handling joint limits, but only on tree joints; we extend this to also handle limits on closure joints by constructing a penalty function to drive limit-exceeding closure DoF back into compliance. No penalty is assigned to in-limits DoF, but upon exceeding a limit the penalty increases quadratically.

Let m be the total number of closure DoF with set limits, and let Π_m be a $[m \times d]$ binary projection matrix selecting the m limited DoF from the d total closure joint DoF. Let $\mathbf{l}_c$ and $\mathbf{u}_c$ be vectors of the m lower and upper closure DoF limits, respectively. The limit excess function is

$$\text{using } \mathbf{v}^T = (v_0, \dots, v_{m-1}), \mathbf{l}^T = (l_0, \dots, l_{m-1}), \mathbf{u}^T = (u_0, \dots, u_{m-1})$$

$$\text{EXCESS}(\mathbf{v}, \mathbf{l}, \mathbf{u}) = (s_0, \dots, s_{m-1})^T \text{ s.t. } s_i = \begin{cases} v_i - u_i & \text{if } v_i > u_i \\ 0 & \text{if } u_i \geq v_i \geq l_i \\ v_i - l_i & \text{if } v_i < l_i, \end{cases} \quad (26)$$

and the *closure limit residual* vector $\mathbf{e}_m$ is the square of the excess for each DoF, computed by the *forward limit error mapping* FK_m :

$$\text{FK}_m : \mathbb{R}^f \rightarrow \mathbb{R}^m \quad \mathbf{e}_m = \text{FK}_m(\mathbf{x}) = \text{DIAG}(\mathbf{s}_c) \mathbf{s}_c \quad (27)$$

with $\mathbf{s}_c = \text{EXCESS}(\Pi_m \Pi_{fc} \text{FK}(\mathbf{x}), \mathbf{l}_c, \mathbf{u}_c)$.

The *limit Jacobian* is the $[m \times f]$ matrix of partial derivatives of 27:

$$J_m = \frac{\partial \mathbf{e}_m}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} \text{FK}_m(\mathbf{x}) = 2 \text{DIAG}(\mathbf{s}_c) \Pi_m \Pi_{fc} J_y. \quad (28)$$

By inserting the limit level just below the (highest-priority) invariant level, the solver will enforce joint limits to the greatest extent possible without violating the DoI of any closure joints, and target and posture waypoints will only be pursued to the extent possible without exceeding any DoF limits.

The final *lock* level enables freezing the pose of selected DoF, with priority between target tracking and limiting. Locked closures are driven to hold the pose they had when originally locked, in an approach similar to target tracking. Locked tree joints are removed from the problem (considered rigid).

Let Γ be an ordering of the set of locked joints, partitioned into sub-sequences of tree and closure joints,

$$\Gamma \subset J = (\Gamma_t, \Gamma_c) \text{ with } \Gamma_t = \Gamma \cap T \text{ and } \Gamma_c = \Gamma \cap C. \quad (29)$$

Let k be the total number of DoF of joints in Γ , and let k_t and k_c be same for Γ_t and Γ_c , so $k = k_t + k_c$. Finally, let $\mathbf{k}$ be a vector of the values of the k_c locked closure DoF at time of locking. A $[k_c \times d]$ binary projection matrix Π_k selects the

k_c locked DoF out of all closure DoF, and the *lock residual* vector $\mathbf{e}_k$ is computed by the *forward lock error mapping* FK_k

$$\text{FK}_k : \mathbb{R}^f \rightarrow \mathbb{R}^k \quad \mathbf{e}_k = \text{FK}_k(\mathbf{x}) = \Pi_k \Pi_{fc} \text{FK}(\mathbf{x}) - \mathbf{k}. \quad (30)$$

The *lock Jacobian* is the $[k_c \times f]$ matrix of partial derivatives of 30:

$$J_k = \frac{\partial \mathbf{e}_k}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} \text{FK}_k(\mathbf{x}) = \Pi_k \Pi_{fc} J_y. \quad (31)$$

4.4 Interacting with Model State

We already briefly presented a comprehensive set of operations for evolving the *structure* of a linkage model (table 2). Corresponding operations to mutate *state* are relatively simple because the heavy lifting—propagating the effects of the mutations—is done by the PDLS action for differential control.

Forward kinematic state change operations simply change the target or posture values for joint DoF. Again, for inverse kinematics, just apply the changes to a general (6-DoF) closure joint attached between the link to be manipulated and the ground link. Rather than set joint DoF individually, it is helpful to provide more intuitive means such as

- *click-and-drag* manipulation—the operator selects links or joints with the mouse and drags them. Because the mouse only supports two axes of translation, attention needs to be paid when the manipulated object has more than two DoF. For example, different mouse buttons and/or keyboard modifiers can be used to select the DoF to operate. Computationally, drag gestures are discretized and used to repeatedly set new target waypoints for the manipulated DoF.
- *special purpose input devices*—mouse-based manipulation is general-purpose and requires no special hardware, but can only control two axes of motion simultaneously. This limitation can be addressed, for example, by providing application-specific input devices with mobility that more closely matches that of the manipulated linkage model. One example of such a device is our TRACK apparatus for ATHLETE (Figure 12).
- *trajectory following*—here the operator specifies a script for the motion of a set of DoF, for example, by entering a sequence of waypoints and the time intervals that should elapse from one to the next. The system then effects the motion by successively generating target waypoints along the specified trajectory.

5 Operating High-DoF Robots

We now turn to the application of the mixed real/virtual interface in operating high-DoF robots. The core idea is to use virtual articulations to (1) constrain and (2) parameterize the intended motion (Figures 1 and 2). The operator is presented with a graphical model of the robot and a catalog of available virtual joints (Figure 5). To constrain motion for a particular task, the operator instantiates joints from this catalog and interconnects them to the links of the actual robot—and/or to newly instantiated virtual links—thereby constructing arbitrary virtual extensions to the actual robot kinematics. Virtual joints can also parameterize specific task DoF; for

example the long prismatic virtual joint in Figure 13 parameterizes the length of a trenching motion.

The system computes local assembly motions as virtual articulations are constructed. From there, the operator can move any joint or link, and the system interactively solves the differential control problem in real-time to find a compatible motion for all joints which best satisfies all constraints. For example, in the trenching task, the operator can effectively command “trench from -0.9m to +0.4m” by operating the corresponding virtual prismatic joint, or they may simply drag the constrained end effector with the mouse. The motions can be validated in simulation and then executed on the hardware, as we have done for ATHLETE (Section 5.2).

This approach to high-DoF operations is generally applicable to kinematic operations in articulated robots of any topology, including both open- and closed-chains as well as both over- and under-constraint. Section 5.3 demonstrates this topology independence in an example with a modular robot. Two layers of structure abstraction are used in this case, which help organize the motion specification problem for the operator, and which also enable the system to hierarchically decompose the full structure into smaller independently solvable parts (Section 4.1).

Two key challenges in high-DoF operations are handling under- and over-constraint; these are discussed next, followed by the ATHLETE and self-reconfiguring robot examples

5.1 Handling Under- and Over-Constraint

Under-constrained systems are especially common in high-DoF robots because they are often used in tasks which involve only a few degrees of constraint, whereas there may be many more degrees of freedom available in the robot. Unlike some prior approaches, which are aimed towards fully constraining the motion [19] (sometimes referred to as “redundancy resolution” [4]), our system allows the operator to add as much or as little constraint as desired.

At the lowest level, the least-squares PDLs solver gives a basic ability to find reasonable motions in any under-constrained system: at each iteration a shortest step is taken in joint space resulting in incrementally minimal motion. Said another way, at a fine scale, the system will produce piecewise linear moves from one configuration to the next in joint space. A more roundabout trajectory might also be feasible, but would likely be surprising.

On top of this foundation, the operator may construct chain-closing virtual articulations to express specific motion constraints, and thus reduce redundancy as much as desired. Figure 2 gives a basic demonstration, though this technique is pervasive in the our approach and is used in all examples.

Over-constraint, where the feasible configuration space is actually empty, is also a possibility. While at first this might seem to be a serious issue, in the presence of virtual articulations, it may be allowable—virtual closure joints can be broken if necessary. Our system also permits over-constraint, and provides the operator with tools to handle it at two levels.

The lowest-level handling of over-constraint is again given by the least-squares PDLs. If any individual priority level is over-constrained, then the least-squares solution will minimize error across all constraints in the level. This is useful and can produce intuitive behavior from the operator’s perspective. For example, when

multiple closure joints need to be broken, the amount of breakage can be balanced across them all.

The PDLS priority levels (Section 4.3) provide a higher-level tool for handling over-constraint: satisfaction of constraints at a lower priority level will not compromise satisfaction at higher levels, even when the constraints conflict. Consider the object inspection task in Figure 12. In this case we use our mimicking hardware interface *TRACK* (Tele-Robotic ATHLETE Controller for Kinematics) to pose the limb holding the inspection camera. But there is also a virtual spherical joint constraining the camera. *TRACK* has no haptic feedback, so while the operator should generally try to pose it near to a feasible configuration, invariably this will diverge from the strict spherical constraint surface, over-constraining the limb. The spherical joint constraint is modeled at the invariant level, and *TRACK*'s pose is modeled at the target level, so the system will sacrifice the latter for the former. The overall effect is as if the virtual spherical joint was physically present and rigidly constraining the motion, with an elastic connection between the pose of *TRACK* and the pose of the actual limb.

While under- and over-constraint is allowable, physical systems that are under-actuated or or inconsistently over-constrained are still possible. Essentially, to the extent that the under- or over-constraint can be restricted to just the virtual articulations, they can be controlled as desired. Many robots are used in fully-actuated tasks and are thus not physically under-constrained; the examples studied in this chapter all fall into this category. While such systems can still be over-constrained, if they can be assembled then at least the over-constraint is self-consistent.

5.2 Experiments with ATHLETE

The All-Terrain Hex-Legged Extra-Terrestrial Explorer is under development for use in future Lunar missions, where it will potentially aid human explorers in various ways [41]. Figure 10 shows two instances of the current hardware, along with its representation in our system. *ATHLETE* weighs about 1000kg and has six identical limbs, each with six revolute joints and a terminal wheel. All joints have harmonic drivetrains and active-off brakes. The limbs attach to a hexagonal deck which is about 2.5m in (circumscribed) diameter and about 1.8m above the ground in nominal pose (foreground in the figure).

The six limbs of the robot are intended both for locomotion (both rolling and walking) and also for manipulation and inspection tasks. Other researchers are considering the locomotion problems [17, 34]; here, we focus on manipulation and inspection. The original operations interface for these kinds of motions, based on traditional approaches, consisted of three primitives:

MOVE_JOINTS—Essentially equivalent to a board with 36 knobs, each commanding the position of one joint (forward kinematic control).

MOVE_TOOL—inverse kinematics (IK) for one leg (body locked in space).

MOVE_BODY—IK for the body (wheels locked in space).

While *MOVE_TOOL* is obviously useful, because it can only control one leg at a time, any task requiring multi-limb motion would need to be coordinated by some additional mechanism. Both *MOVE_JOINTS* and *MOVE_BODY* can move all the limbs together, but the former has obvious shortcomings (it simply puts the full burden of motion specification on the operator), and the latter is limited to controlling body posture alone.

Even in the case of single-limb motion, MOVE_TOOL is not always ideal. For example, some tasks do not constrain all 6 DoF of the EE pose; consider a drilling task—rotation of the EE about the drill axis may be unconstrained. And there are often multiple discrete solutions for fully constrained poses, i.e., ATHLETE’s “elbows” can generally kink in different directions to reach the same EE pose. Thus, as an initial contribution, we built a direct-manipulation input device that mimics one ATHLETE limb, called the Tele-Robotic ATHLETE Controller for Kinematics (TRACK) [22] (Figure 12, inset). But this does not solve the whole problem—used alone, it still provides no mechanism for constraining motion or for coordinating motion of multiple limbs.

Virtual articulations can aid the operation of specific whole-robot motions which are rapid for human operators to conceptualize but difficult to express in more traditional interfaces. We implemented five different task scenarios; four on the hardware [36] and one in simulation.

- Figure 11 (simulation): A leg-mounted camera is used to inspect a crew module in a cylindrical scanning motion, with a virtual prismatic joint to control elevation and a virtual revolute joint for radial angle.
- Figure 12 (hardware): a limb-mounted camera inspects a roughly spherical object while maintaining a constant distance
- Figure 13 (hardware): a trench is inspected, with the support legs moving the deck to extend reachable trench length
- Figure 14 (hardware): a fixed camera is made to pan and tilt with the motion both parameterized and constrained by virtual revolute joints
- Figure 15 (hardware): two limbs execute a pinching maneuver with the distance and angles controlled by virtual prismatic and revolute joints.

For the object inspection task the operator models the camera distance constraint using a virtual spherical joint between the object (represented as a virtual link) and the camera. The space of reachable viewpoints is extended, relative to moving only the camera limb alone, by using the five other limbs to lean the hexagonal deck. Because the deck often carries a payload, it should remain flat; this is expressed by a virtual Cartesian-3 joint between the deck and the world frame. After configuring these virtual articulations the operator can drag the camera with the mouse to scan the object.

Descriptions of the other experiments are similar. For the bi-manual experiment the robot was partially supported by an overhead crane, as simultaneously raising two limbs is not supported on the current hardware. The crane served as a safety-backup in the other experiments.

5.3 Simulation of a Modular Robot

Moving to a second example in high-DoF operations, we now show a simulated modular robot with over 100 joints, demonstrating both the topology independence and scalability of our system. Two layers of structure abstraction are applied in this example, breaking up the operator’s motion specification task and also enabling hierarchical decomposition for efficient motion computation.

The system in this section is a *self-reconfiguring* modular robot. Modular SR robots, studied since at least the late 1980s [15], are systems which can globally form arbitrary shapes by re-arranging many smaller interconnected units [23,30,31].

This is another class of robot where large numbers of joints are possible—though hardware has so far been limited to a few 10s of modules, work has been done in simulation with larger numbers of up to 1 million [13].

The SR system we use consists of active barbell-shaped modules with two symmetric rotating distal grippers and passive bars [9]. This system is capable of arbitrary topological reconfiguration by attaching and detaching modules; we have previously shown a reconfiguration sequence that builds the tower we study here from a disassembled set of parts [8].

As a *chain-type* modular robot, this system is also capable of *deformation* motions where the inter-module connectivity remains constant, but modules cooperatively use their internal kinematic DoF to effect a global shape-change. Operating this type of motion—i.e. providing an interface where an operator can conveniently specify general deformations in potentially large modular assemblies—has been under-explored; using virtual articulations to constrain and parameterize coordinated motion is a natural fit. By their nature, SR systems can assume arbitrary topologies, and one of the strong points of the virtual articulation approach is topology independence. Structure abstraction (Section 4.1) is also particularly useful because repeated sub-structures are common in large SR constructions. (A related idea in topological reconfiguration is the concept of *meta-modules* [35].)

Our main result is interactive operation of a tower with 120 simulated robot joints and about 150 virtual joints (Figure 17). The tower is built from a repeating block sub-structure (Figure 16). The operator first defines a particular constrained motion at the block level, explained in the next section, and this induces the full-tower motions presented in the following section. Structure abstraction plays a key role, and enables a hierarchical decomposition which keeps an otherwise sluggish motion computation snappy. (Williams and Mahew [42] considered a similar scenario, but that work was for one particular structure, whereas our algorithms are general-purpose.)

As illustrated in Figure 16, the operator wants the tower block to act as if it had two overall DoF: left/right tilting, and up/down expansion and contraction. A highest-level abstraction replaces the block with a chain of one revolute and one prismatic joint. But the actual modules making up the block—the implementation of this abstraction—have more complexity. They form two legs, which can be considered as four-bar linkages. Zooming in to a single leg, the desired motion is like a piston: the end-to-end distance of the constituent chain of modules sets the length of the leg, and the endpoints can also pivot to allow the leg to rotate. This implies a second level of abstraction where each leg is virtually replaced by a piston-like assembly of virtual prismatic and revolute joints. Finally, a Cartesian-2 constraint is added to keep the middle link of each leg parallel with the axis of the leg. In this case, the abstractions can be kept proper (Section 4.1) by sufficiently limiting the range of motion of the joints in the interface linkages.

It is important to note that these particular motion constraints, and this particular set of structure abstractions, are merely the designs of the operator in this instance. Other constraints and abstractions are possible: the idea is that the operator conceptualizes a desired motion, and then expresses it by designing virtual articulations and structure abstractions.

Once the operator is satisfied with the operation of a single block, a number of them are attached together to give towers of varying height. This final assembly is done at the highest level of abstraction, so that the top-level linkage is simply

a linear chain of alternating revolute and prismatic joints along the “backbone” of the tower. (This is similar to the backbone-curve method for controlling hyper-redundant robots [6].) The operator can then interactively specify a motion, e.g. by click-and-drag interaction with the mouse, for any link or joint in the top-level linkage. The system will solve the differential control problem in and respond with a deformation as shown in Figure 17.

In this example, structure abstractions help the operator in the same way that traditional algorithmic abstraction aids the architect of a large software system: they break the problem up by enforcing high-level semantic invariants on sub-systems. The designer (in either situation) is thus not required to address the whole detailed problem at once.

These structure abstractions will also result in a hierarchical decomposition (Section 4.1) with three PDL rounds (Section 4.2): First, the backbone will move as the operator has specified. Second, the top-level abstraction for each block will drive the mid-level abstractions for each leg. Finally, those abstractions will drive the motion of the modules that comprise the legs. The first round will consist of a single PDL call, and the size of the system will scale linearly as the tower grows in height. The later rounds are different: there, growing the tower will result in a larger number of independent PDL calls, but the system size in each call will remain constant.

In fact, the tower should move in a very similar way if constructed as a single flat (non-hierarchical) linkage. But in this case there would be only one PDL round; the size of the system would scale linearly with tower height, but the constant factors would be greater than in the hierarchical case, making a significant difference in computation time. To check this, we conducted an experiment with towers of varying heights, up to 15 blocks, built in four different ways. The resulting measured computation times are comparatively plotted in Figure 17. (Each test was run interactively on a lightly loaded workstation.) The “plain” tower used no hierarchy, and also omitted the constraining Cartesian-2 joints. The “constrained” tower added the constraining joints, and the “hierarchical” tower added the structure abstractions, but not the constraining joints. The “constrained hierarchical” tower included both the abstractions and the constraining joints.

In each case adding the Cartesian-2 constraints increases the computation time, and adding the structure abstractions decreases it. The most interesting comparison is between the “constrained” tower (the worst case) and “constrained hierarchical”: the latter remains roughly on-par with the un-constrained versions, at about 20ms per simulation iteration, which is acceptable for interactive response. But all the additional closure joints slow down the system for the non-hierarchical “constrained” tower by forcing PDL of large systems; performance degrades to over 100ms per iteration for a 15 block tower, making interactive response sluggish.

6 Conclusions

This chapter has presented the theory and implementation of a novel operations system for articulated robots with large numbers (10s to 100s) of DoF, based on virtual articulations and kinematic abstractions. Operating such robots is a challenging problem because they are capable of many different kinds of motion, but often this requires coordination of large numbers of joints. Prior methods exist for specifying motions at both low and high-levels of detail; the new method fills a gap

in the middle by allowing the operator to be as detailed as desired in specifying the motion, and is independent of any particular robot or task topology.

The main idea is to allow the operator to dynamically add, remove, and interconnect virtual joints and links with a model of the real robot. Virtual joints can both parameterize task motion and can constrain coordinated motion by closing kinematic chains. Virtual links can represent task-relevant coordinate frames, and also serve as intermediate connection points for constructions involving multiple virtual joints.

For topologically large robots, structure abstraction is a kinematic analogue to the algorithmic abstractions commonly used in software systems. This allows the operator to break the motion specification task into smaller sub-problems, and also enables the system to compute motions more rapidly via hierarchical decomposition.

Our approach does have some limitations: speed and force commanding are not directly addressed; and a large class of trajectories will be constructible even given a limited catalog of joints, some will be out of reach. For example, a true helical joint cannot be synthesized from the current joint catalog. Non-holonomic constraints and rolling contact are also not possible.

Portions of the work described in this chapter were funded under the NSF EFRI program and by a Caltech/NASA/JPL DRDF grant. We thank Daniela Rus, David Mittman, Jeff Norris, and Brian Wilcox for helping to make this project possible. We also thank the RSVP team at NASA/JPL for providing a VRML model of ATHLETE.

References

1. P. Baerlocher and R. Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *Visual Computer*, 20:402–417, 2004.
2. Paolo Baerlocher. *Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures*. PhD thesis, EPFL, 2001.
3. H. Bruyninckx. Open RObot Control Software (OROCOS). www.orocos.org.
4. Herman Bruyninckx. *Kinematic Models for Robot Compliant Motion with Identification of Uncertainties*. PhD thesis, Katholieke Universiteit Leuven, 1995.
5. P. Chiacchio, S. Chiaverini, L. Sciavicco, and B. Siciliano. Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy. *IJRR*, 10(4):410–425, 1991.
6. G. Chirikjian and J. Burdick. The kinematics of hyper-redundant robot locomotion. *IEEE Trans. on Robotics and Automation*, 11(6):781–793, 1995.
7. E. Davis. Approximation and abstraction in solid object kinematics. Technical Report TR1995-706, NYU Computer Science, 1995.
8. C. Detweiler, M. Vona, K. Kotay, and D. Rus. Hierarchical control for self-assembling mobile trusses with passive and active links. In *IEEE International Conference on Robotics and Automation*, pages 1483–1490, 2006.
9. C. Detweiler, M. Vona, Y. Yoon, S. Yun, and D. Rus. Self-assembling mobile linkages. *IEEE Robotics and Automation Magazine*, 14:45–55, 2007.
10. Antonio Diaz-Calderon, Issa A. D. Nesnas, Hari Das Nayar, and Won S. Kim. Towards a unified representation of mechanisms for robotic control software. *International Journal of Advanced Robotic Systems*, 3(1):061–066, 2006.

11. L. Dobrjanskyj and F. Freudenstein. Some applications of graph theory to the structural analysis of mechanisms. *ASME Journal of Engineering for Industry*, pages 153–158, February 1967.
12. R. Featherstone and D. Orin. Robot dynamics: Equations and algorithms. In *IEEE ICRA*, pages 826–834, 2000.
13. Robert Fitch and Zack Butler. Million module march: Scalable locomotion for large self-reconfiguring robots. *International Journal of Robotics Research*, 27(3/4):331–343, 2008.
14. Lorenzo Flückiger. A robot interface using virtual reality and automatic kinematics generator. In *Int. Symposium on Robotics*, pages 123–126, April 1998.
15. Toshio Fukuda and Seiya Nakagawa. Dynamically reconfigurable robotic system. In *IEEE ICRA*, pages 1581–1586, 1988.
16. Michael L. Gleicher. *A Differential Approach to Graphical Interaction*. PhD thesis, Carnegie Mellon University, School of Computer Science, 1994.
17. Kris Hauser, Timothy Bretl, Jean-Claude Latombe, and Brian Wilcox. Motion planning for a six-legged lunar robot. In *WAFR*, pages 301–316, 2006.
18. Oleg Ivlev and Axel Gräser. An analytical method for the inverse kinematics of redundant robots. In *Proceedings of 3rd ECPD Int. Conf. on Advanced Robots, Intelligent Automation and Active Systems*, pages 416–421, 1997.
19. Oleg Ivlev and Axel Gräser. Resolving redundancy of series kinematic chains through imaginary links. In *Proceedings of CESA '98 IMACS Multiconference, Computational Engineering in Systems Applications*, pages 477–482, 1998.
20. Evangelos Kokkevis. Practical physics for articulated characters. In *Game Developers Conference*, 2004.
21. Alain Liégeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-7(12):868–871, December 1977.
22. D. Mittman, J. Norris, M. Powell, R. Torres, C. McQuin, and M. Vona. Lessons Learned from All-Terrain Hex-Limbed Extra-Terrestrial Explorer Robot Field Test Operations at Moses Lake Sand Dunes, Washington. In *AIAA Space*, 2008.
23. Mark Moll and Daniela Rus (eds.). Special issue on self-reconfiguring modular robots. *International Journal of Robotics Research*, 27(3/4), March/April 2008.
24. S. Nakaoka, A. Nakazawa, K. Yokoi, H. Hirukawa, and K. Ikeuchi. Generating whole body motions for a biped humanoid robot from captured human dances. In *IEEE ICRA*, pages 3905–3910, 2003.
25. Cary B. Phillips, Jianmin Zhao, and Norman I. Badler. Interactive real-time articulated figure manipulation using multiple kinematic constraints. In *Proceedings of SIGGRAPH*, pages 245–250, 1990.
26. J. Pratt, C. Chew, A. Torres, P. Dilworth, and G. Pratt. Virtual model control: An intuitive approach for bipedal locomotion. *IJRR*, 20(2):129–143, 2001.
27. Jerry E. Pratt. Virtual model control of a biped walking robot. Master's thesis, Massachusetts Institute of Technology, 1995.
28. Mitchell Pryor. *Task-Based Resource Allocation for Improving the Reusability of Redundant Manipulators*. PhD thesis, University of Texas at Austin, 2002.
29. Mitchell W. Pryor, Ross C. Taylor, Chetan Kapoor, and Delbert Tesar. Generalized software components for reconfiguring hyper-redundant manipulators. *IEEE/ASME Transactions on Mechatronics*, 7(4):475–478, December 2002.
30. Daniela Rus, Zack Butler, Keith Kotay, and Margette Vona. Self-reconfiguring robots. *Communications of the ACM*, 45(3):39–45, 2002.

31. Daniela Rus and Gregory S. Chirikjian (eds.). Special issue on self-reconfigurable robots. *Autonomous Robots*, 10(1), January 2001.
32. B. Siciliano and J.-J.E. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *Fifth International Conference on Advanced Robotics*, pages 1211–1216, 1991.
33. Russel Smith. Open dynamics engine, 2008. <http://www.ode.org>.
34. T. Smith, J. Barreiro, D. Smith, V. SunSpiral, and D. Chavez-Clemente. ATHLETE’s Feet: Multi-Resolution Planning for a Hexapod Robot. In *ICAPS*, 2008.
35. S. Vassilvitskii, J. Kubica, E. Rieffel, J. Suh, and M. Yim. On the general reconfiguration problem for expanding cube style modular robots. In *IEEE ICRA*, pages 801–808, 2002.
36. M. Vona, D. Mittman, J. Norris, and D. Rus. Using virtual articulations to operate high-DoF manipulation and inspection motions. In *FSR*, 2009.
37. Marsette Vona. Hierarchical decomposition and kinematic abstraction with virtual articulations. In *Advances in Robot Kinematics*, pages 33–43, 2010.
38. Marsette Vona. *MSim: Mixed Real/Virtual Simulator and Interface*, 2011. <http://www.ccs.neu.edu/research/gpc/msim>.
39. Marsette A. Vona. *Virtual Articulation and Kinematic Abstraction in Robotics*. PhD thesis, EECS, Massachusetts Institute of Technology, August 2009.
40. C. Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master’s thesis, Simon Fraser University, 1993.
41. B. Wilcox, T. Litwin, J. Biesiadecki, J. Matthews, M. Heverly, J. Morrison, J. Townsend, N. Ahmad, A. Sirota, and B. Cooper. ATHLETE: A cargo handling and manipulation robot for the moon. *Field Robotics*, 24:421–434, 2007.
42. R. Williams and J. Mayhew. Control of truss-based manipulators using virtual serial models. In *ASME DETC*, 1996.
43. Giles D. Wood and Dallas C. Kennedy. Simulating mechanical systems in simulink with SimMechanics. Technical report, The Mathworks, 2003.
44. K. E. Zanganeh and J. Angeles. A formalism for the analysis and design of modular kinematic structures. *IJRR*, 17(7):720–730, 1998.

Fig. 1. Operating ATHLETE with virtual articulations.

A virtual revolute joint (circled) is interactively added by an operator to facilitate tilting the field of view of one of ATHLETE's cameras, which are rigidly mounted to the robot (the camera is not visible in the model). Camera tilt can then be commanded by rotating the virtual joint; feasible whole-robot supporting motions are automatically computed on-line. Figure 14 shows this experiment on the actual hardware. ATHLETE model courtesy of the RSVP team, NASA/JPL/Caltech.

Fig. 2. Structure abstraction and kinematic constraint.

Virtual articulations can be used both for *structure abstraction* and to constrain intended motion. The actual robot mechanism in (a) is an open chain of four revolute joints. The operator is designing a piston-like behavior of the endpoints. In our system, this can be modeled as a revolute-prismatic-revolute construction (b) which virtually replaces the model of the actual mechanism (c,d), forming an abstraction. Left alone, the middle link remains underconstrained (e); the operator restricts it by adding an assembly of two virtual prismatic joints (f) below the abstraction barrier. The same constraint can also be defined using a shorthand *Cartesian-2* joint (g).

Fig. 3. Kinematic graph (*linkage*) examples.

Left: a mixed real/virtual model with superimposed linkage graph; right: a second example. Links (rigid bodies) correspond to vertices and joints correspond to edges (such graphs are not novel). We always identify a spanning tree (dark edges) with the remaining *closure joints* (light edges) closing kinematic cycles.

Fig. 4. Overall architecture of our implementation.

level l	height h_l	residual e_l	Jacobian J_l
invariant	i	e_i (Eq. 11)	J_i (Eq. 14)
limit	m	e_m (Eq. 27)	J_m (Eq. 28)
lock	k_c	e_k (Eq. 30)	J_k (Eq. 31)
target	t	e_t (Eq. 17)	J_t (Eq. 18)
posture	p_c	e_{pc} (Eq. 22)	J_{pc} (Eq. 23)

Table 1. The five SOLVE priority levels in order from highest to lowest.

Fig. 5. 3D joints available in our implementation.

Each joint type is represented graphically in our interface system. Connections to links are shown as Bézier curves.

Fig. 6. The forward kinematic mapping.

The forward kinematic mapping FK computes the *closure state* $\mathbf{y}$ as a function of the *tree state* $\mathbf{x}$. $\mathbf{y}$ is a concatenated vector of the closure joint $(\mathbf{t}, \boldsymbol{\theta})$ state vectors; $\mathbf{x}$ is a concatenated vector of the DoF of the tree joints.

Fig. 7. Hierarchical linkages and structure abstraction.

Left: sub-linkage L_1 is driving, L_2 is simultaneous, and L_3 is driven. Tree joints are dark colored arrows; closure joints are light. Right: an example of structure abstraction. The “implementation” A , a series chain of four revolute joints, is virtually replaced by a simpler piston-like “interface” I . Also see Figure 2.

primitive name	description	primitive name	description
ADDLINK	add new link	SETTYPE	change joint type
SETLINKNAME	change name	SETLIMITS	change joint limits
ADDJOINT	add new joint	INVERT	invert a joint
SETJOINTNAME	change name	MAKEGROUND	switch ground link
MAKETREE	change disposition	REPOSITIONLINK	move link
MAKECLOSURE	change disposition	REPOSITIONJOINT	move joint
SETPARENT	re-attach a joint	SPLIT	insert new link and joint
SETCHILD	re-attach a joint	MERGE	delete joint and child link
REMOVEJOINT	delete a joint	ENCAPSULATE	demarcate sub-linkage
REMOVEDLINK	delete a link	DISSOLVE	merge sub-linkage

Table 2. The structure mutation primitives.

Fig. 8. The local assembly problem.

A virtual revolute joint has been added, closing a chain between the “ankles” of two adjacent ATHLETE limbs. The chain is initially inconsistent (a) because the pose that was specified for the new joint relative to the limbs differs from the actual current relative pose. A red curve indicates the error. The system solves the local assembly problem by moving the legs to remove the error, as shown in (b) and (c). Once the new topology is assembled, it can be operated (d) as originally intended.

Fig. 9. The differential control problem.

Here the operator uses click-and-drag interaction with the mouse to interactively pose the last link in an ATHLETE limb. The hex deck is locked, so the system solves the differential control problem corresponding to the single-limb inverse kinematics.

Fig. 10. All-Terrain Hex-Legged Extra-Terrestrial Explorer (ATHLETE). Left: hardware (image courtesy NASA/JPL/Caltech); right: model.

Fig. 11. Simulation of ATHLETE inspecting a cylindrical crew module.

Fig. 12. ATHLETE inspecting an object with a leg-mounted camera.

A leg-mounted camera inspects an object from a range of views, while a virtual spherical joint centered on the object maintains a fixed focal distance. Our system automatically computes postural motions that greatly extend the range of reachable viewpoints vs. actuating a single leg alone. Also, for this task our mimicking hardware interface device *TRACK* was combined with the virtual articulation system.

Fig. 13. ATHLETE performing a trenching motion.

A scoop is mounted and could be used to dig a trench (digging would also require force control). Virtual prismatic joints constrain and parametrize the trench length and depth, virtual revolute joints set the scoop angle, and a Cartesian-3 joint keeps the deck at a constant orientation (e.g. in case that the deck is carrying an additional payload). A similar trenching motion could be operated by inverse kinematics for the active limb alone, but the addition of constrained postural motion approximately doubles the total possible trench length.

Fig. 14. Pan/tilt of an ATHLETE camera via postural motions.

ATHLETE has a number of side-facing cameras rigidly mounted to its deck (no pan/tilt actuators). Postural motions, where all legs cooperatively rotate the deck about the camera's center of focus, can be used to aim them. This can be operated with two crossed-axis virtual revolute joints. With this construction in place, quantified commands—"pan 30 degrees left, tilt 10 degrees down"—have a specific meaning. Our system finds a corresponding whole-robot motion.

Fig. 15. ATHLETE performing bi-manual manipulation motions.

In this bi-manual operations experiment, a virtual link models a movable object that could be grasped by pinching between two adjacent wheels (grasping a physical object would require the addition of force control). The operator can drag this virtual link around in our interface; the system finds compatible kinematic motions for the legs. Motion is automatically limited to the reachable workspace by constraint prioritization—the joint closure constraints are higher priority than the object pose target. Virtual prismatic and revolute joints additionally parametrize the grip.

Fig. 16. Operating a tower block with two levels of structure abstraction.

The actual modules comprising a tower block can move in a variety of ways, but the operator intends only two motions: the block should tilt left/right and expand up/down, forming the highest-level abstraction (left subfigure). A secondary constraint is that each 4-bar leg should effectively act like a piston (c.f. Fig. 2), so the legs are further constrained and abstracted (middle and right subfigures; c.f. Fig. 7).

Fig. 17. Operating deformations in 5- and 15-block towers.

Blocks (Fig. 16) are stacked to make towers; a 15-block tower has 271 total joints. 120 represent real joints, the remaining implement virtual constraints and structural abstractions. Here the operator is dragging the top of the tower and the system interactively solves the differential control problem to deform the tower. The block-level abstractions both aid motion design and enable hierarchical decomposition, keeping computation time below 20ms/iteration. A similar tower operated without abstractions required over 100ms (“constrained” vs “constrained hierarchical”).