URDF+: An Enhanced URDF for Robots with Kinematic Loops

Matthew Chignoli[1], Jean-Jacques Slotine[1], Patrick M. Wensing[2], and Sangbae Kim[1]

***Abstract*— Designs incorporating kinematic loops are becoming increasingly prevalent in the robotics community. Despite the existence of dynamics algorithms to deal with the effects of such loops, many modern simulators rely on dynamics libraries that require robots to be represented as kinematic trees. This requirement is reflected in the de facto standard format for describing robots, the Universal Robot Description Format (URDF), which does not support kinematic loops resulting in closed chains. This paper introduces an enhanced URDF, termed URDF+, which addresses this key shortcoming of URDF while retaining the intuitive design philosophy and low barrier to entry that the robotics community values. The URDF+ keeps the elements used by URDF to describe open chains and incorporates new elements to encode loop joints. We also offer an accompanying parser that processes the system models coming from URDF+ so that they can be used with recursive rigid-body dynamics algorithms for closed-chain systems that group bodies into local, decoupled loops. This parsing process is fully automated, ensuring optimal grouping of constrained bodies without requiring manual specification from the user. We aim to advance the robotics community towards this elegant solution by developing efficient and easy-to-use software tools.**

I. Introduction

The recursive dynamics algorithms upon which modern rigid-body dynamics (RBD) libraries are built [1], [2] were initially developed only for open-chain systems. To date, these libraries [3], [4], [5], [6], [7] have not implemented techniques for dealing with kinematic loops that are as simple or efficient as the original recursive algorithms. Instead, they resort to either (i) approximating their dynamic effects or (ii) using non-recursive algorithms that scale poorly with the robot's dimension. This lack of attention given to kinematic loops likely contributed to the decision made by the original developers of the Universal Robotic Description Format (URDF) [8] not to support the modeling of robots with kinematic loops. Despite lacking such support, the URDF has become the de-facto standard format for describing robot models [9].

With designs involving kinematic loops becoming increasingly popular (Fig. 1) as a means to achieve proximal actuation [10], this shortcoming is no longer acceptable. Designs such as parallel belt transmissions [11], differential drives [12], [10], [13], and four-bar mechanisms [14], [13], [15] enable high-speed limb motion while focusing the inertia of the actuators in the robot's base structure.

While the original recursive dynamics algorithms were developed for open chains, they can be adapted to systems

[1]Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA: `chignoli@mit.edu`

[2]Department of Aerospace and Mechanical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA

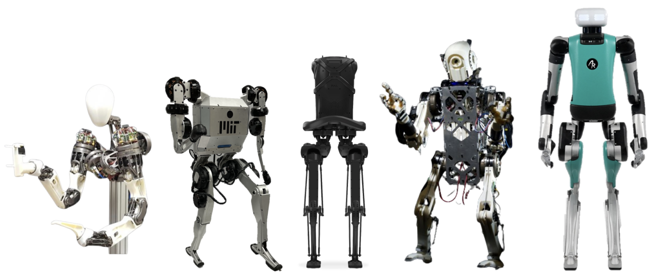

Fig. 1. Robots employing kinematic loops to achieve proximal actuation. Left to right: LIMS2-AMBIDEX [21], MIT Humanoid [11], Kangaroo [15], Hermes [12], Digit [22].

with kinematic loops. This was first recognized by Jain, who approached the problem through Spatial Kernel Operators (SKO) [16] and demonstrated that systems with kinematic loops can be represented with SKO models [17]. Thus, they are compatible with the original recursive algorithms [18]. In our recent work [19], we provided a self-contained derivation of these algorithms from Featherstone's perspective of propagation methods [4], which is the prevailing perspective among modern RBD libraries [3], [5], [6], [7].

These "constraint-embedding" algorithms for closed chains have yet to achieve widespread proliferation. One possible reason may be a lack of efficient, easy-to-use software tools employing these techniques. The goal of this paper, along with our related RBD library [20], is to push the robotics community toward embracing this elegant solution for dealing with a critical problem facing the field.

However, we want to emphasize that our push does not involve a paradigm shift away from the URDF. Many commendable attempts at larger-scale shifts have been proposed [23]. For example, the Simulation Description Format (SDF) [24] offers many of the features requested by URDF users [25], such as support for multiple robots, support for several types of sensors, and support for kinematic loops. MuJoCo's MJCF format [7] uses a kinematic-tree-based design philosophy similar to URDF's. The MJCF supports new and more detailed elements compared to URDF, such as sensors, actuators, constraints, and contacts. The Asynchronous Multi-Body Framework (AMBF) Format [26], on the other hand, uses an entirely different modular design philosophy aimed at improving human readability and constraint handling.

Despite these efforts, a majority of the community believes that URDF will be more commonly used in the future [25]. Therefore, we accept this sentiment and choose to augment, rather than replace, the URDF. The main features of our augmented format, the URDF+, are:

- Simple additions to the original URDF data structures that allow for many more constraints to be modeled,
- A new parser that automatically produces models whose bodies are optimally grouped according to Jain's minimal aggregation criteria [27].

When the kinematic loops of the system are "local," i.e., involve a small number of bodies, the optimal grouping makes the parsed model well-suited for constraint-embedding algorithms [17], [19]. In cases where the loops are not local and non-recursive methods such as [28] are more efficient, the optimal parsing is still useful in providing the sparsity pattern of the constraint Jacobians. The URDF+ data structures and parser are implemented open-source [29], [30] as forks of the ROS URDF parser [31], [32].

In this work, we also provide examples demonstrating how URDF+ can model complicated closed-chain robots incompatible with the existing URDF format. We emphasize that the URDF+ retains the design philosophy of the URDF with which so many in the community are familiar. URDF+ files are fully backward-compatible with URDF. RBD libraries and simulators can either (i) update their algorithms to use the fully recursive techniques of [17], [33], [19] or (ii) keep their existing algorithms and use the new description format to model and compute closed-loops using their existing non-recursive algorithms.

The rest of the paper is organized as follows. Sec. II provides background on robot modeling, the URDF, and recursive algorithms for closed-chain systems. Sec. III and IV describe how URDF+ modifies the data structures and parser of the existing URDF, respectively. Examples of how the URDF+ is employed are shown in Sec. V. Finally, Sec. VI concludes the work and discusses future steps toward achieving efficient and accessible dynamic simulation for robotics.

II. Background

A. System Modeling

Robotic systems are commonly modeled using graphs [4], [34]. A graph $\mathcal{G}$ consists of a set of nodes $\mathcal{N}$ and undirected edges $\mathcal{E}$. When the graph $\vec{\mathcal{G}}$ is directed (also called a digraph), its edges $\vec{\mathcal{E}}$ are directed from one node to another. A graph describing a robotic system is called a connectivity graph (CG) and has the following properties:

- The nodes represent bodies.
- The edges represent joints.
- Exactly one root node represents a fixed base.
- The graph is undirected and connected.

A graph is called a topological tree when exactly one path exists between any two nodes in a graph. A spanning tree (ST) of a CG is a subgraph containing all of the original CG nodes along with a set of edges in the original CG such that the subgraph is a topological tree. These included edges constitute the set of tree joints $\mathcal{T}$. The leftover edges constitute the set of loop joints $\mathcal{L}$. Thus, for a CG with N_B non-root nodes and N_J edges, there will be N_B tree joints and $N_L = N_J - N_B$ loop joints. We will describe connectivity graphs by their bodies, tree joints, and loop joints, $\mathcal{G}_C = (\mathcal{B}, \mathcal{T} \cup \mathcal{L})$.

The properties of spanning trees are used to develop the "regular numbering" convention for assigning identifying numbers to the nodes and edges [4]:

1) Choose the edges to include in the ST.
2) Assign the number 0 to the root node.
3) Assign the remaining nodes from 1 to N_B so each node has a higher number than its parent in the ST.
4) Number the edges in the ST such that edge i connects node i to its parent.
5) Number all remaining edges from $N_B + 1$ to N_J in any order.

We will use the following index convention to distinguish between the different types of joints. The indices i and j will be used to index tree joints and bodies (1 to N_B), l will be used to index loops (1 to N_L), and k identifies the loop joint that closes loop l ($k = l + N_B$).

The number of tree joint variables, n, and loop-closure constraints, n^c, are given by

$$n = \sum_{i=1}^{N_B} n_i, \quad n^c = \sum_{k=N_B+1}^{N_J} n_k^c, \tag{1}$$

where n_i is the degrees of freedom permitted by the ith tree joint, and n_k^c is the number of constraints imposed by the kth loop joint.

We now consider the particular case where the original CG is an ST. Such a CG is called a "kinematic tree" and corresponds to a robot free of kinematic loops. As previously noted, the current URDF can only represent robots as kinematic trees.

B. Joint Models

A robot's configuration can be described by the poses of the coordinate frames attached to each of its bodies. We use the following convention to describe a coordinate frame: $F_{i,j}$. The subscript i denotes which body the frame is rigidly attached to. The subscript j denotes which joint the frame is associated with. When $i = j$, we omit j, leaving F_i. A schematic of these coordinate frames and their relationships is shown in Fig. 2.

Two spatial transforms are used to transform from $F_{\lambda(i)}$ to F_i, where $\lambda(i)$ is the parent of body i. The tree transform $\mathbf{X}_{T(i)}$ is a fixed transform that describes the pose of $F_{\lambda(i),i}$ relative to $F_{\lambda(i)}$. This intermediate frame $F_{\lambda(i),i}$ gives the location of F_i when $\mathbf{q}_i = \mathbf{0}$. The joint transform $\mathbf{X}_{J(i)}$ gives the pose of F_i relative to $F_{\lambda(i),i}$ for arbitrary $\mathbf{q}_i$. The joint transform is a function of the joint position $\mathbf{q}_i$ and depends on the joint type.

For the kth loop joint, the predecessor transform $\mathbf{X}_{P(k)}$ is a fixed transform that gives the pose of $F_{p(k),k}$ relative to $F_{p(k)}$, where $p(k)$ is the predecessor body. The successor transforms $\mathbf{X}_{S(k)}$ does the same for the successor body. Finally, the joint transforms $\mathbf{X}_{J(k)}$ describes the transform from $F_{p(k),k}$ to $F_{s(k),k}$.

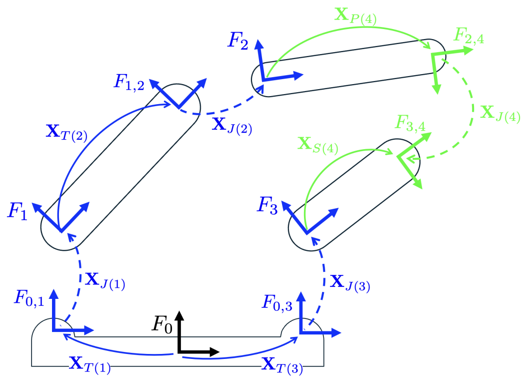

Fig. 2. Exploded view of a four-bar mechanism showing its coordinate frames and the transforms between them. Tree joint quantities are shown in blue, and loop joint quantities are shown in green.

Joint models provide the information to describe the permitted relative motion between connected bodies. This information is captured by three quantities: $\mathbf{X}_J$, $\mathbf{S}$, and $\mathbf{\Psi}$. The motion subspace matrix $\mathbf{S}_i \in \mathbb{R}^{6\times n_i}$ maps the joint velocity $\dot{\mathbf{q}}_i$ to the difference in 6D spatial velocity between the preceding and succeeding bodies. Similarly, the constraint force subspace matrix $\mathbf{\Psi}_i \in \mathbb{R}^{6\times n_i^c}$ maps the constraint forces $\mathbf{f}_i^c$ to the 6D spatial force across the joint. The joint model determines how to compute these quantities given the joint position $\mathbf{q}_i$.

C. Loop Constraints

The motion constraints imposed by loop joints can be expressed in either "implicit" form

$$\phi(\mathbf{q}) = 0, \quad \mathbf{K}\dot{\mathbf{q}} = 0, \quad \mathbf{K}\ddot{\mathbf{q}} = \mathbf{k}, \tag{2}$$

or, in cases where an independent set of coordinates exists, explicit form

$$\mathbf{q} = \gamma(\mathbf{y}), \quad \dot{\mathbf{q}} = \mathbf{G}\dot{\mathbf{y}}, \quad \ddot{\mathbf{q}} = \mathbf{G}\ddot{\mathbf{y}} + \mathbf{g}, \tag{3}$$

where $\mathbf{q}$ is the set of complete coordinates of the robot and $\mathbf{y}$ is the set of independent coordinates. The constraint Jacobians and biases for the implicit and explicit constraints are given by

$$\mathbf{K} = \frac{\partial \phi(\mathbf{q})}{\partial \mathbf{q}}, \quad \mathbf{k} = -\dot{\mathbf{K}}\dot{\mathbf{q}}, \quad \mathbf{G} = \frac{\partial \gamma(\mathbf{y})}{\partial \mathbf{y}}, \quad \mathbf{g} = \dot{\mathbf{G}}\dot{\mathbf{y}}.$$

D. URDF

The main idea behind the URDF is that it encodes a kinematic tree. The description of the entire CG is contained within the contents of the `<robot>` element. The nodes of the graph (links of the robot) are given by the `<link>` elements. Similarly, the edges of the graph (tree joints of the robot) are described by `<joint>` elements. The contents of the `<link>` elements describe the dynamics and appearance of the respective links. For example, the `<inertial>` child element gives the link's spatial inertia, and the `<visual>` child element provides information about its appearance (shape, size, color, etc.). Similarly, the contents of `<joint>` elements describe kinematic constraints between links. The `<parent>` and `<child>` elements give the links being constrained, the `<origin>` gives the tree transform $\mathbf{X}_T$, and the `type` and `<axis>` describe the joint model. For more information on the elements and attributes comprising URDF files, see [8].

E. Recursive Algorithms for Closed-Chain Systems

The critical insight to enable recursive algorithms for closed chains, as originally demonstrated in [17] and revisited in [33], [19], is the transformation of non-ST connectivity graphs into ST connectivity graphs via the grouping of bodies involved in local loop constraints. Grouping the bodies enables loop constraints to be resolved locally, i.e., only when that group of bodies is encountered during a forward or backward pass. This local treatment avoids the need for large-scale matrix factorization. The original presentation of the algorithms [17] refers to the grouping as "constraint embedding" and the resulting groups of bodies as "aggregate links." In service of our propagation method-based perspective [19], we previously used the terms "clustering" and "clusters," respectively. We default to the original nomenclature (constraint embedding and aggregate links) here due to less reliance on it in our subsequent development.

In modeling robots as graphs, constraint embedding involves representing multiple bodies as a single node. Specifically, the bodies constituting an aggregate link are represented with a single node. When all bodies are grouped in their respective aggregate nodes, the resulting connectivity graph is guaranteed to be an ST and is referred to as a loop-aggregated connectivity graph (LACG). A LACG $\mathcal{G}_A$ consists of the aggregate links $\mathcal{C}$ and the collections of tree and loop joints associated with each group of bodies $\mathcal{T}_C$. Following this process, constraints are embedded within the aggregate link and will not otherwise lead to loop constraints with other groups. A key property of the recursive algorithms for closed chains is that their advantage over non-recursive methods diminishes as the size of the aggreate links increases [35], [19]. Thus, while the choice of aggregate links may be non-unique, there always exist a subset of optimal groupings. In a thorough, graph-theoretic-based analysis of multibody system dynamics, Jain derives a criteria for minimal aggregation that, when satisfied, guarantees the model is optimally grouped [27].

III. Modifications to Description Format

We first address the challenge of extending the description format of the URDF to accommodate kinematic loops. In addressing this challenge, we want to be mindful of preserving the properties of the URDF that have led to its proliferation. Specifically, the current URDF is intuitive, has a low barrier to entry, and is compatible with many software interfaces. To preserve these properties, we ensure that URDF+:

1) Uses elements that correspond to physically meaningful properties,
2) Requires no knowledge of constraint embedding,
3) Minimally modifies the existing URDF,
4) Maintains backward compatibility with the URDF.

URDF+ makes only three modifications to the URDF. Two are new child elements of the `<robot>` element, and one is an optional new attribute of the `<joint>` element. The key idea behind our modification is that we maintain all of the elements URDF uses to describe kinematic trees and instead use them to describe *spanning* trees. We then use our new elements to encode the loop joints and complete the connectivity graph. We emphasize that the user does not have to specify the aggregate links manually. Aggregation takes place “under the hood” from the user's perspective, which is important since requiring detailed knowledge of spatial kernel operators, constraint embedding, or clustering could raise the barrier to entry considerably.

A. *Loops*

The first element we add to the URDF to make URDF+ is the `<loop>` element. As noted earlier, loops refer to edges present in the CG but absent from the ST. Most CGs permit multiple STs. Thus, it is the responsibility of the user to determine which joints to declare as tree joints and which to declare as loop joints. Some choices are more natural than others (i.e., declaring the controlled and observed joints as tree joints), but all combinations are supported by URDF+.

The information needed to specify a `<loop>` is similar to that for a `<joint>`, although slightly more information is required. The following template shows the full contents of the `<loop>` element:

```
<loop name="name" type="type">
    <predecessor name="name"/>
        <origin xyz="x y z" rpy="r p y"/>
    </predecessor>
    <successor name="name"/>
        <origin xyz="x y z" rpy="r p y"/>
    </successor>
    <axis xyz="x y z"/>
</loop>
```

For the kth loop joint, the `<predecessor>` element gives the predecessor node $p(k)$ via the `name` attribute and the predecessor transform $\mathbf{X}_{P(k)}$ via the `<origin>` child element. The `<successor>` element provides the same information but for the successor node $s(k)$. Like the `<joint>` element, the `type` attribute and `<axis>` child element specify the joint model.

The `<loop>` element described above contains all the information needed to formulate the implicit constraint associated with the lth loop. The first step is to find the tree joints involved in the loop constraint. This is done by finding the nearest common ancestor (NCA) between the predecessor and successor. The “ancestors” of rigid body i are all the rigid bodies in the ST that are on the path from the root body to body i. We use $j \preceq i$ to denote that body j is an ancestor of body i, which we emphasize is different from $j \leq i$. Thus, the NCA between predecessor and successor is given by:

$$\text{nca}\left((p(k), s(k)\right) = \max\left\{i \,|\, i \preceq p(k),\, i \preceq s(k)\right\}. \tag{4}$$

The tree joints involved in the lth loop are those encountered on the path from the NCA to the predecessor and those encountered from the NCA to the successor. These sets of joints are the tree joints associated with the bodies in the path subchains $\nu_{p(k)}$ and $\nu_{s(k)}$, respectively. These path subchains are defined by,

$$\begin{aligned} \nu_{p(k)} &= p(k) \cup \{i \,|\, \text{nca}\left((p(k), s(k)\right) \prec i,\, i \preceq p(k)\}, \\ \nu_{s(k)} &= s(k) \cup \{i \,|\, \text{nca}\left((p(k), s(k)\right) \prec i,\, i \preceq s(k)\}. \end{aligned} \tag{5}$$

The standard definition of $\mathbf{K}_l$ has its jth block column as [4]

$$\mathbf{K}_{lj} = \epsilon_{lj} \mathbf{\Psi}_k^\top \mathbf{S}_j, \tag{6}$$

where ϵ_{lj} is -1 for $j \in \nu_{p(k)}$, 1 for $j \in \nu_{s(k)}$, and 0 for all other j. For conciseness, we omit from (6) the spatial transforms that ensure $\mathbf{\Psi}_k$ and $\mathbf{S}_j$ are expressed in the same frame. Recall that since the joint model is known for loop l, $\mathbf{\Psi}_k$ is known, and since the joint models are known for all spanning joints, all $\mathbf{S}_j$ are known.

To prepare for later constraint grouping operations, we will remove all columns where $\epsilon_{lj} = 0$ so that $\mathbf{K}_l$ has n_k^c rows and n_ν columns instead of n columns, where

$$n_\nu = \sum_{i \in \nu_p} n_i + \sum_{i \in \nu_s} n_i. \tag{7}$$

The algorithmic steps for computing (6) are given in Algorithm 8.4 of [4].

B. *Coupling Constraints*

Joints encode kinematics constraints on the relative motion between connected bodies. While joints capture many motion constraints encountered in robotics, they do not capture another popular type of constraint: coupling constraints. Coupling constraints enforce relationships between joint states $\mathbf{q}$ rather than between motions of rigid bodies. In other words, they *couple* the constraint imposed by one joint with the constraint imposed by a different joint. Therefore, they cannot be described by conventional joint transforms, motion subspaces, and constraint force subspaces. We instead make a new element, `<coupling>`, to describe such constraints.

For now, we restrict the class of possible coupling constraints to linear relationships between the positions of tree joints. The following template shows the full contents of the `<coupling>` element:

```
<coupling name="name">
    <predecessor name="name"/>
    <successor name="name"/>
    <ratio value="value"/>
</coupling>
```

For the coupling constraint represented as the kth loop joint, the `<predecessor>` and `<successor>` elements again give $p(k)$ and $s(k)$. The coupling constraint also depends on the NCA of the predecessor and successor nodes. Specifically, the `<ratio>` element gives the ratio between (i) the position of the predecessor tree joint relative to the NCA and (ii) the rotation of the successor tree joint relative to the NCA[1],

$$\sum_{i \in \nu_{p(k)}} \mathbf{q}_i = \eta_k \sum_{j \in \nu_{s(k)}} \mathbf{q}_j \tag{8}$$

[1]The `<joint-mimic>` element that exists for the conventional URDF is a specific case of `<coupling>` where the NCA of the predecessor and successor is also the parent of both bodies.

Note that (8) requires all of the tree joints associated with the bodies in ν_p and ν_s to have the same number of degrees of freedom and to encode the same type of motion (i.e., rotation or translation).

Unlike `<loop>` constraints, which tend to be most naturally represented as implicit constraints, coupling constraints are most naturally represented as explicit constraints, where the explicit constraint Jacobian $\mathbf{G}_k$ is a function of only η_k. A geared transmission is the most common example of a coupling constraint, although many variations exist in robotics.

C. Independent Coordinates

If the user wishes to represent any of the loop constraints in explicit form, they must also specify a set of independent generalized velocity coordinates $\dot{\mathbf{y}}$. We remark that the recursive algorithms for closed-chain systems only require the loop constraint Jacobians and biases, $\mathbf{G}_k$ and $\mathbf{g}_k$, to be expressed explicitly. They do not rely on explicit constraint definitions of the form $\mathbf{q} = \gamma(\mathbf{y})$. Therefore, even in cases where loop constraints are formulated as implicit (e.g., (6)), equivalent explicit constraint Jacobians and biases can be systematically derived [33].

The choice of independent coordinates is non-unique, so the final modification we make to the URDF is an optional attribute allowing the user to specify which tree joints should be included in the independent coordinates and which should not. We name this attribute `independent` and show a template for its usage here:

```
<joint name="name" type="type" independent="bool">
    <origin xyz="x y z" rpy="r p y"/>
    <axis xyz="x y z"/>
    <parent name="name"/>
    <child name="name"/>
</joint>
```

Since the attribute is optional, the model will successfully parse if omitted. However, in that case, the user will be restricted to describing its configuration using valid sets of ST coordinates. By making the attribute optional, we satisfy our goal of not requiring the user to have knowledge of constraint embedding.

The URDF+ method of determining independent coordinates restricts the independent coordinates to be a subset of the complete spanning coordinates ($\dot{\mathbf{y}} \subseteq \dot{\mathbf{q}}$), even though the recursive algorithms for closed-chain systems permit alternative choices. Furthermore, we note that for a model with n degrees of freedom, the number of independent coordinates is $n^i = n - \sum_{l=1}^{N_L} \text{rank}\,(\mathbf{K}_l)$. To deal with cases where the user specifies an incompatible number, we have ensured that the parser will detect the incompatibbility and throw an error.

IV. Parser

The new data structures encoded by the URDF+ necessitate an accompanying new parser. The parser for the original URDF processed the elements encoding nodes and edges and produced a kinematic tree. However, because the URDF+ supports closed chains, the new parser must perform additional processing to produce a loop-aggregated connectivity graph. This parser, specifically its automation of Jain's constraint embedding strategy [27], constitutes the second contribution of this work.

A. Relation to Strongly Connected Components

Jain's constraint embedding strategy [27] requires the user to "identify the smallest subtree that contains [the predecessor and successor bodies of a loop constraint]." This identification is not trivial and, especially in cases of nested or overlapping loops, may require background knowledge of constraint embedding - which we aim to avoid. Therefore, we propose a parser that automatically identifies the minimal aggregation links. We do this by using the concept of strongly connected components in a directed graph. First, consider the physical interpretation of the minimal aggregation criteria: an aggregate link $\mathcal{C}$ is minimally aggregated if and only if for every pair of bodies $(B_i, B_j) \in \mathcal{C}$, a valid motion of B_i cannot be computed without simultaneously computing the motion of B_j, and vice versa. This condition parallels the definition of a Strongly Connected Component (SCC) in graph theory [36]: a strongly connected component of a digraph $\vec{\mathcal{G}} = (\mathcal{N}, \vec{\mathcal{E}})$ is a maximal set of vertices $\mathcal{V} \subset \mathcal{N}$ such that for all $V_i, V_j \in \mathcal{V}$, V_i is reachable from V_j and V_j is reachable from V_i.

We base our URDF+ parser on this parallel. Specifically, our parser consists of three steps, shown in Alg. 1. We first create the CG by reading the URDF file, which is similar to the original URDF Parser and shown in Alg. 2. Next, we build a directed graph where the SCCs in the digraph correspond to minimally aggregated links. We call this digraph a "constraint dependency digraph." Finally, we use a standard SCC algorithm [36] to extract the SCCs from the constraint dependency digraph (CDD) and, therefore, form the LACG.

Algorithm 1 URDF+ Parser

```
Require: robot.urdf
1: $\mathcal{G}_C$ = connectivityGraphFromUrdf (robot.urdf)
2: $\vec{\mathcal{G}}_D$ = constraintDependencyDigraphFromCG ($\mathcal{G}_C$)
3: $\mathcal{G}_A$ = extractStronglyConnectedComponents ($\vec{\mathcal{G}}_D$)
4: return $\mathcal{G}_C, \mathcal{G}_A$
```

B. Constraint Dependency Digraph

Forming a constraint sub-group is not as simple as concatenating the ν_p and ν_s for a given loop joint. For example, consider the cases of "nested" and "overlapping" loops, shown in Fig. 3(a). A nested loop occurs when the predecessor or successor of a loop joint l is in the path subchain $\nu_{p(k')}$ or $\nu_{s(k')}$ of another loop joint l'. Overlapping loops occur when a single body is the predecessor or succesor of multiple loop joints l and l'. In both of these cases, the motions of the bodies in $\nu_{p(k)}$ and $\nu_{s(k)}$ must be simultaneously computed with those of the bodies in $\nu_{p(k')}$ and $\nu_{s(k')}$.

Dealing with these cases, therefore, requires extra steps. The idea behind the CDD is to use the path subchains, which

Algorithm 2 connectivityGraphFromUrdf

```
Require: robot.urdf
1: $\mathcal{B} = \{\}$, $\mathcal{T} = \{\}$, $\mathcal{L} = \{\}$
2: for every <link> in robot.urdf do
3:     Create body $B$ from <link>
4:     $\mathcal{B} \leftarrow \mathcal{B} \cup B$
5: end for
6: for every <joint> in robot.urdf do
7:     Build tree joint $T$ from <joint>
8:     $\mathcal{T} \leftarrow \mathcal{T} \cup T$
9: end for
10: for every <loop> and <coupling> in URDF+ do
11:     Build loop joint $L$ from <loop> or <coupling>
12:     $\mathcal{L} \leftarrow \mathcal{L} \cup L$
13: end for
14: return $\mathcal{G}_C \leftarrow (\mathcal{B}, \mathcal{T} \cup \mathcal{L})$
```

are easy to find, to construct a directed graph and then allow the reachability between nodes in the digraph to determine the aggregation. The aggregation links emerging from applying SCC decomposition to the CDD are guaranteed to satisfy the minimal aggregation criteria. The steps for building the CDD are given by Alg. 3 as well as depicted in Fig. 3.

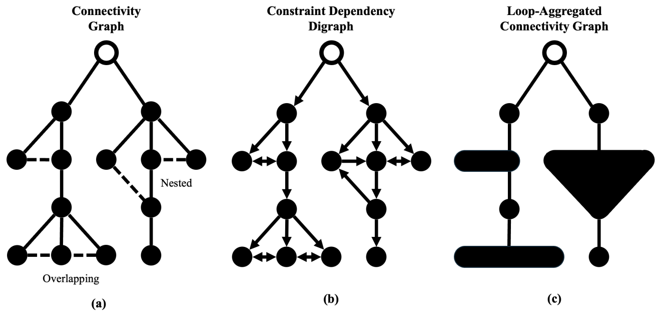
Connectivity Graph
Constraint Dependency Digraph
Loop-Aggregated Connectivity Graph
Nested
Overlapping
(a)
(b)
(c)

Fig. 3. Illustrative connectivity graphs for a system with kinematic loops resulting in multiple aggregate links.

The constraint dependency digraph contains all the nodes in the original connectivity graph. Every tree joint edge in the original spanning tree is then added to the digraph as a directed edge from its parent to its child. On the other hand, every loop joint leads to two directed edges in the constraint dependency digraph. These two edges capture the dependence between the motion of the predecessor and the successor subchain and the motion of the successor and the predecessor subchain. The first directed edge goes from the predecessor to the lowest numbered node in ν_s, and the second directed edge goes from the successor to the lowest numbered node in ν_p. Thus, the constraint dependency digraph has N_B nodes and $N_J + N_L$ directed edges.

C. Extracting Sub-Groups

Standard algorithms in computer science exist for automatically decomposing graphs in SCCs [36]. The algorithm we use involves two depth-first searches, one on the CDD

Algorithm 3 constraintDependencyDigraphFromCG

```
Require: $(\mathcal{B}, \mathcal{T}, \mathcal{L})$
1: $\mathcal{N} = \mathcal{B}$, $\vec{\mathcal{E}} = \{\}$
2: for every tree joint $T_i \in \mathcal{T}$ do
3:     Create directed edge $\vec{E}$ from $B_{\lambda(i)}$ to $B_i$
4:     $\vec{\mathcal{E}} \leftarrow \vec{\mathcal{E}} \cup \vec{E}_i$
5: end for
6: for every loop joint $L_i \in \mathcal{L}$ do
7:     $B_{nca} = \mathrm{nca}\left((p(L_i), s(L_i)\right)$
8:     $\nu_p = \mathrm{pathSubchain}\left(p(L_i), B_{nca}\right)$
9:     $\nu_s = \mathrm{pathSubchain}\left(s(L_i), B_{nca}\right)$
10:    Create directed edge $\vec{E}_i$ from $p(L_i)$ to $\min \nu_s$
11:    Create directed edge $\vec{E}_j$ from $s(L_i)$ to $\min \nu_p$
12:    $\vec{\mathcal{E}} \leftarrow \vec{\mathcal{E}} \cup \{\vec{E}_i, \vec{E}_j\}$
13: end for
14: return $\vec{\mathcal{G}}_D \leftarrow \left(\mathcal{N}, \vec{\mathcal{E}}\right)$
```

Algorithm 4 pathSubchain

```
Require: $B_i, B_j$
1: $\nu = \{\}$
2: while $B_i \neq B_j$ do
3:     $\nu \leftarrow \nu \cup B_i$
4:     $B_i \leftarrow \lambda(B_i)$
5: end while
6: return $\nu$
```

and another on its reverse digraph. The reverse digraph is formed by flipping the direction of every edge in the original digraph. Upon sorting the sub-groups, the parser stores the corresponding constraints $\mathbf{K}_l$ and $\mathbf{G}_l$ with the correct sub-groups, and the newly formed LACG is ready to be used in recursive dynamics algorithms. Note that even though our parser gives an optimal sub-grouping, some of them may still be large due to the robot's morphology. In these cases, sparsity-exploiting algorithms (e.g., [37], [28]) may outperform recursive ones. The URDF+ data structures and parser are still useful because in the process of parsing the model, the original spanning tree is stored, and so too are the loop constraints $\mathbf{K}$ and $\mathbf{G}$, along with their sparsity patterns.

V. Examples

We provide some illustrative examples to demonstrate how to use the new features of URDF+ and how to apply them to model different types of closed-chain mechanisms. Note that for the sake of space, we only include the portions of the URDF+ that are needed to communicate how to use the new features. Therefore, some examples might be "underspecified" and not directly usable by an RBD library.

A. LIMS2-AMBIDEX Wrist

We first provide an example using the `<loop>` element, applying it to a 2-DOF virtual rolling joint for wrist pitch/roll motion [21]. A picture of the joint is shown in Fig. 4. The clever design of the mechanism allows it to emulate the pure rolling contact of two spheres while maintaining a

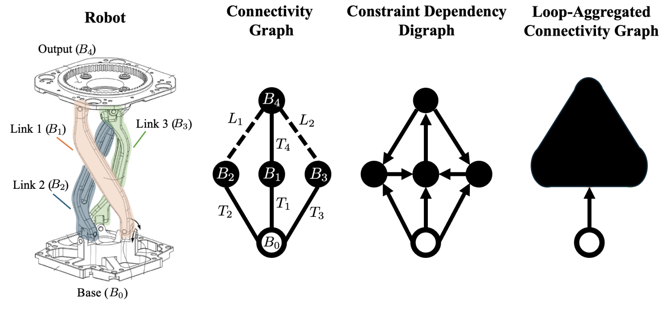

Fig. 4. Schematic of the 2-DOF wrist joint for the LIMS2-AMBIDEX robot [21] and its corresponding connectivity graphs.

wide range of motion free of singular poses. The connecting rods create kinematic loops that prevent the mechanism from being accurately modeled by URDF.

The abridged URDF+ for this mechanism is:

```
<link name="Base"/>
<link name="Link 1"/>
<link name="Link 2"/>
<link name="Link 3"/>
<link name="Output"/>
<joint type="universal" independent="true">
    <parent name="Base"/>
    <child name="Link 1"/>
</joint>
<joint type="universal" independent="false">
    <parent name="Base"/>
    <child name="Link 2"/>
</joint>
<joint type="universal" independent="false">
    <parent name="Base"/>
    <child name="Link 3"/>
</joint>
<joint type="universal" independent="false">
    <parent name="Link 1"/>
    <child name="Output"/>
</joint>
<loop type="universal">
    <predecessor name="Link 2"/>
    <successor name="Output"/>
</loop>
<loop type="universal">
    <predecessor name="Link 3"/>
    <successor name="Output"/>
</loop>
```

The connectivity graph, the constraint dependency digraph, and the loop-aggregated connectivity graph for this URDF+ are also shown in Fig. 4. Observe in the constraint dependency digraph that the connecting rods and output body are all reachable from one another. Thus, they constitute an SCC. Following the parsing rules from Sec. IV, the implicit constraint Jacobian for the mechanism is

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 \\ \mathbf{K}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{\Psi}_1^\top \mathbf{S}_1 & \mathbf{\Psi}_1^\top \mathbf{S}_2 & \mathbf{0} & \mathbf{\Psi}_1^\top \mathbf{S}_4 \\ \mathbf{\Psi}_2^\top \mathbf{S}_1 & \mathbf{0} & \mathbf{\Psi}_2^\top \mathbf{S}_3 & \mathbf{\Psi}_2^\top \mathbf{S}_4 \end{bmatrix} \tag{9}$$

Since the total number of degrees of freedom of the tree joints is 8, and the rank of the constraint matrix is 6, the mechanism has the expected number of independent degrees of freedom: 2.

B. Parallel Belt Transmission

We also provide an example of a coupled constraint as part of a parallel belt transmission [11], shown in Fig. 5. The simplest example of a coupled constraint is a geared

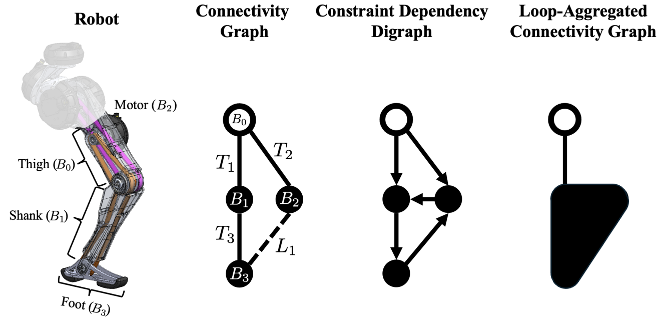

Fig. 5. CAD view of the parallel belt transmission in the MIT Humanoid [11] and its corresponding connectivity graphs.

transmission. This is easily modeled by URDF+. However, there are also simple changes to the conventional recursive algorithms that can approximately account for the effects of geared motors [38]. For a more complicated design like a parallel belt transmission where the NCA of the predecessor and successor is not also the parent of the predecessor and successor, the constraint gets more complicated, and the geared motor approximation technique fails to generalize.

The abridged URDF+ for this mechanism is:

```
<link name="thigh"/>
<link name="shank"/>
<link name="motor"/>
<link name="foot"/>
<joint type="revolute" independent="true">
    <parent name="thigh"/>
    <child name="shank"/>
</joint>
<joint type="revolute" independent="false">
    <parent name="thigh"/>
    <child name="motor"/>
</joint>
<joint type="revolute" independent="true">
    <parent name="shank"/>
    <child name="foot"/>
</joint>
<coupling>
    <predecessor name="foot"/>
    <successor name="motor"/>
    <ratio value="eta"/>
</coupling>
```

The connectivity graph, the constraint dependency digraph, and the loop-aggregated connectivity graph for this URDF+ are also shown in Fig. 5. The predecessor subchain ν_p includes the foot and shank, and the successor subchain ν_s includes only the motor. Therefore, the explicit constraint can be expressed

$$\mathbf{q}_{\text{shank}} + \mathbf{q}_{\text{foot}} = \eta \mathbf{q}_{\text{motor}}. \tag{10}$$

leading to the constraint Jacobian

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\mathbf{q}}_{\text{shank}} \\ \dot{\mathbf{q}}_{\text{foot}} \\ \dot{\mathbf{q}}_{\text{motor}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1/\eta & 1/\eta \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}_{\text{shank}} \\ \dot{\mathbf{q}}_{\text{foot}} \end{bmatrix} = \mathbf{G}\dot{\mathbf{y}}. \tag{11}$$

VI. Conclusion

The introduction of URDF+ represents an advancement in the ability of modern RBD libraries to seamlessly support robots with kinematic loop designs. By enhancing the conventional URDF to include such loops while preserving its intuitive design and usability, URDF+ addresses a critical gap in the existing standards. Our modifications maintain the familiar elements of URDF for describing kinematic trees and add new elements for loop joints, ensuring compatibility with recursive algorithms for closed-chain systems. The development of an automated parser to handle the new elements and generate loop-aggregated connectivity graphs underscores the user-centric approach of URDF+. This automation eliminates the need for manual specification of aggregate links, simplifying the modeling process for users and ensuring optimal performance. Through illustrative examples and the creation of supporting tools, we demonstrate the practical benefits and feasibility of URDF+. Our goal is to encourage the robotics community to adopt this enhanced format, which will facilitate the design and simulation of more complex robotic systems. Future work will focus on adding features such as more complicated coupling constraints (e.g. differential joints), specifying arbitrary independent coordinates, and parsing models in a manner that detects when which constraints should be handled with constraint-embedding versus non-recursive alternatives such as [28] or [39].

References

[1] D. E. Orin, R. McGhee, M. Vukobratović, and G. Hartoch, "Kinematic and kinetic analysis of open-chain linkages utilizing newton-euler methods," *Mathematical Biosciences*, vol. 43, no. 1-2, pp. 107–130, 1979.

[2] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *The international journal of robotics research*, vol. 2, no. 1, pp. 13–30, 1983.

[3] M. L. Felis, "Rbdl: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, pp. 1–17, 2016.

[4] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.

[5] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019.

[6] J. Carpentier, F. Valenza, N. Mansard, *et al.*, "Pinocchio: fast forward and inverse dynamics for poly-articulated systems." https://stack-of-tasks.github.io/pinocchio, 2021.

[7] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[8] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System.* " O'Reilly Media, Inc.", 2015.

[9] D. Tola and P. Corke, "Understanding urdf: A dataset and analysis," *IEEE Robotics and Automation Letters*, 2024.

[10] Y. Sim and J. Ramos, "Tello leg: The study of design principles and metrics for dynamic humanoid robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9318–9325, 2022.

[11] M. Chignoli, D. Kim, E. Stanger-Jones, and S. Kim, "The mit humanoid robot: Design, motion planning, and control for acrobatic behaviors," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pp. 1–8, 2021.

[12] A. Wang, J. Ramos, J. Mayo, W. Ubellacker, J. Cheung, and S. Kim, "The hermes humanoid system: A platform for full-body teleoperation with balance feedback," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 730–737, 2015.

[13] Y. Liu, J. Shen, J. Zhang, X. Zhang, T. Zhu, and D. Hong, "Design and control of a miniature bipedal robot with proprioceptive actuation for dynamic behaviors," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 8547–8553, 2022.

[14] T. Apgar, P. Clary, K. Green, A. Fern, and J. W. Hurst, "Fast online trajectory optimization for the bipedal robot cassie.," in *Robotics: Science and Systems*, vol. 101, p. 14, Pittsburgh, Pennsylvania, USA, 2018.

[15] A. Roig, S. K. Kothakota, N. Miguel, P. Fernbach, E. M. Hoffman, and L. Marchionni, "On the hardware design and control architecture of the humanoid robot kangaroo," in *6th workshop on legged robots during the international conference on robotics and automation (ICRA 2022)*, 2022.

[16] A. Jain, "Unified formulation of dynamics for serial rigid multibody systems," *Journal of Guidance, Control, and Dynamics*, vol. 14, no. 3, pp. 531–542, 1991.

[17] A. Jain, "Recursive algorithms using local constraint embedding for multibody system dynamics," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 49019, pp. 139–147, 2009.

[18] A. Jain and G. Rodriguez, "Recursive dynamics for geared robot manipulators," in *29th IEEE Conference on Decision and Control*, pp. 1983–1988, 1990.

[19] M. Chignoli, N. Adrian, S. Kim, and P. M. Wensing, "A propagation perspective on recursive forward dynamics for systems with kinematic loops," *arXiv preprint arXiv:2311.13732*, 2024.

[20] Robotics, Optimization, and Assistive Mobility Lab @ Notre Dame, "Generalized rigid body dynamics algorithms," 2024.

[21] H. Song, Y.-S. Kim, J. Yoon, S.-H. Yun, J. Seo, and Y.-J. Kim, "Development of low-inertia high-stiffness manipulator lims2 for high-speed manipulation of foldable objects," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4145–4151, 2018.

[22] Agility Robotics, "Digit," 2024.

[23] M. Ivanou, S. Mikhel, and S. Savin, "Robot description formats and approaches," in *2021 International Conference" Nonlinearity, Information and Robotics"(NIR)*, pp. 1–5, 2021.

[24] Open-Source Robotics Foundation, "Sdformat," 2024.

[25] D. Tola and P. Corke, "Understanding urdf: A survey based on user experience," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pp. 1–7, 2023.

[26] A. Munawar, Y. Wang, R. Gondokaryono, and G. S. Fischer, "A real-time dynamic simulator and an associated front-end representation format for simulating complex robots and environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1875–1882, 2019.

[27] A. Jain, "Multibody graph transformations and analysis: part ii: closed-chain constraint embedding," *Nonlinear dynamics*, vol. 67, pp. 2153–2170, 2012.

[28] J. Carpentier, R. Budhiraja, and N. Mansard, "Proximal and sparse resolution of constrained dynamic equations," in *Robotics: Science and Systems 2021*, 2021.

[29] mit-biomimetics, "urdfdom headers," 2024.

[30] mit-biomimetics, "urdfdom," 2024.

[31] ros, "urdfdom headers," 2024.

[32] ros, "urdfdom," 2024.

[33] R. Kumar, S. Kumar, A. Müller, and F. Kirchner, "Modular and hybrid numerical-analytical approach-a case study on improving computational efficiency for series-parallel hybrid robots," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3476–3483, 2022.

[34] A. Jain, "Graph theoretic foundations of multibody dynamics: Part ii: Analysis and algorithms," *Multibody system dynamics*, vol. 26, pp. 335–365, 2011.

[35] A. Jain, C. Crean, C. Kuo, and M. B. Quadrelli, "Efficient constraint modeling for closed-chain dynamics," in *The 2nd Joint International Conference on Multibody System Dynamics, Stuttgart, Germany*, Citeseer, 2012.

[36] D. E. Knuth, *The art of computer programming*, vol. 3. Pearson Education, 1997.

[37] R. Featherstone, "Exploiting sparsity in operational-space dynamics," *The International Journal of Robotics Research*, vol. 29, no. 10, pp. 1353–1368, 2010.

[38] R. P. Paul, *Robot manipulators: mathematics, programming, and control: the computer control of robot manipulators*. Richard Paul, 1981.

[39] A. Sathya and J. Carpentier, "Constrained articulated body dynamics algorithms," *hal-04443056*, 2024.