2026 IEEE/SICE International Symposium on System Integration (SII)
January 11-14, 2026. Cancún, México

scikit-robot: An Integrated Framework for Solving Structural Challenges in Dynamic Modeling for Reconfigurable Robotics

Iori Yanokura[1], Kento Matsuo[1], Hirokazu Ishida[1], Sousuke Morita[1], Aoi Nakane[1], Naoya Yamaguchi[1], Tasuku Makabe[1] and Kei Okada[1]

***Abstract*— Modular reconfigurable robots offer a high degree of flexibility through dynamic hardware morphology changes, but their potential has been constrained by software limitations, particularly the static design philosophy of URDF, the de facto standard format. This paper identifies three fundamental research gaps that hinder the development of reconfigurable robots: (1) the disconnect between programmatic "decomposition" of models and runtime "composition," (2) the lack of standard mechanisms for updating kinematic topology at runtime, and (3) the limited semantic expressiveness of the URDF format itself. To address these challenges, we propose scikit-robot, a unified framework that integrates multiple components for reconfigurable robotics. The framework establishes a design-to-deployment workflow through: (1) an integrated URDF toolchain handling decomposition, reconfiguration, and mesh optimization, (2) a dynamic root transformation algorithm enabling flexible physical connections, and (3) a comprehensive hash-based model management system providing asset identity verification and supporting simulation-to-hardware integration. Experimental validation on multiple real modular robot systems demonstrates stable operation and direct model transfer from simulation to physical hardware, confirming the effectiveness of our integrated workflow approach. scikit-robot provides researchers and developers with a practical foundation for dynamic and adaptive robotic systems.**

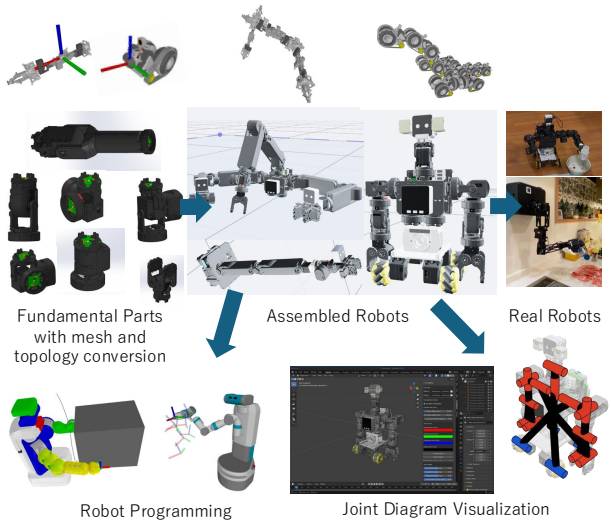

Fig. 1. An overview of the integrated workflow in scikit-robot for modular reconfigurable robotics. The framework facilitates a seamless transition from fundamental parts to simulated assembled robots, and finally to real-world hardware. It encompasses tools for robot programming and kinematic visualization, bridging the gap between digital design and physical deployment.

I. Introduction

The evolution of robotics has long been driven by the pursuit of increasingly adaptive and versatile systems. Among these efforts, modular reconfigurable robots represent an important research direction: robotic systems capable of dynamically altering their physical morphology to adapt to diverse tasks and environments [1]. This approach offers potential advantages by enabling a single robotic platform to perform multiple functions through runtime reconfiguration, potentially reducing costs and increasing operational flexibility across numerous applications [2]. This concept has been explored through various systems ranging from early pioneers like CEBOT [3], PolyBot [4], and M-TRAN [5], to more recent advances including Roombots for adaptive furniture [6], SMORES-EP with parallel self-assembly capabilities [7], and CONCERT for construction applications [8].

However, despite decades of research and significant hardware advances, the widespread adoption of reconfigurable robotics remains limited. While the mechanical foundations for modular systems have matured considerably, challenges persist in the software infrastructure required to support dynamic reconfiguration. This limitation is particularly notable in the realm of robot modeling and description, where existing standards and tools remain primarily static in their design philosophy.

The Unified Robot Description Format (URDF) [9]–[11] has emerged as the de facto standard for robot modeling, providing a common language for describing robot kinematics, dynamics, and visualization across diverse platforms and applications. URDF's widespread adoption within the Robot Operating System (ROS) ecosystem has made it a key tool for modern robotics development. However, URDF was originally conceived for static, fixed-morphology robots, and its design reflects this heritage through immutable hierarchical structures and rigid parent-child relationships that cannot be modified during runtime [12], [13].

This limitation creates challenges for reconfigurable robotics. When robot morphology changes—whether through the addition of new modules, reconfiguration of existing connections, or dynamic topology updates—the entire robot model must be regenerated from scratch. This process is computationally expensive and prone to errors, inconsistencies, and integration challenges that increase with system complexity.

[1]The authors are with the Department of Mechano-Informatics, University of Tokyo, Tokyo, Japan {yanokura, matsuo, ishida, morita, nakane, yamaguchi, makabe,k-okada}@jsk.imi.i.u-tokyo.ac.jp

A. Research Problem Identification

Our analysis reveals three main challenges that collectively form the "reconfigurable robotics software gap". First, current URDF workflows follow a linear, unidirectional path: design → generate → execute. This paradigm breaks down for reconfigurable systems that require iterative cycles of decomposition (breaking complex models into reusable components) and composition (assembling components into new configurations) during runtime execution. A critical manifestation of this problem is that standard tools cannot dynamically modify kinematic root assignments, constraining module design to fixed hierarchical structures rather than flexible physical connection logic. Second, ROS's immutable parameter system prevents dynamic updates to robot models during execution. Once a robot_state_publisher or MoveIt!'s RobotModel [14] is initialized, the kinematic structure becomes frozen, creating incompatibility with reconfigurable robotics requirements. Third, the URDF format has limited semantic expressiveness and lacks the semantic richness necessary to express concepts essential to reconfigurable robotics, such as module interfaces, connection constraints, or compatibility relationships between components.

B. Contributions of This Work

This paper presents scikit-robot, a unified framework that integrates multiple components to address reconfigurable robotics challenges. Our work addresses the gap between simulation-based design and real-world implementation through three interconnected contributions:

1. **Integrated Workflow for Modular URDF Manipulation**: We present a unified framework that encompasses the complete development cycle—from initial model decomposition through composition, mesh optimization, inverse kinematics computation, collision-free motion planning, simulation validation, to direct real robot deployment—within a single Python environment. This reduces the barriers between design tools, simulation environments, and hardware interfaces.

2. **Dynamic Root Transformation Algorithm**: Our algorithm enables non-root link connections by dynamically reconfiguring URDF hierarchical structures. This method removes constraints that have limited modular robot design, expanding the space of achievable configurations.

3. **Hash-based Model Management and Distribution**: We introduce a content hashing system that encompasses not only URDF files but all referenced assets (meshes, textures), serving as the foundation for reliable model management. This system provides model identity verification, enables automatic distribution, and supports simulation-to-hardware transfer.

We demonstrate the effectiveness of our integrated approach through deployment on multiple physical modular robot systems, showing that our framework provides practical tools for real-world reconfigurable robotics applications.

II. Background and Related Work

Recent developments with yourdfpy [15] have begun to address the robust loading of diverse URDFs, building upon earlier Python URDF processing work [16], [17]. Enhanced URDF processing techniques [18] have also contributed to this field. However, the challenge of dynamic reconfiguration remains unresolved. In high-performance libraries like Pinocchio [19], URDF serves merely as an initial input format, with its role being primarily that of an "exchange format" between tools rather than a computational performance consideration.

EusLisp [20] provided rich insights that became the foundation for this framework, but faced challenges in integration with the Python ecosystem. Timor [21] specializes in configuration optimization, modular_hhcm [22] focuses on specific hardware generation, Concert [8] provides collaborative development platforms, and Onshape-to-robot [23] enables CAD integration, but none of them addresses the gap in general-purpose dynamic model manipulation and management that our research aims to fill.

Table I shows that each tool has complementary relationships with different value propositions:

TABLE I
Comparison of Existing Tools by Design Objectives

Tool	Modular URDF	Topology Change	Pure Python	Integrated Toolchain
Timor	✓	×	×	△
modular_hhcm	△	×	✓	×
Concert	△	×	×	△
Onshape-to-robot	✓	×	✓	×
scikit-robot	✓	✓	✓	✓

III. Proposed System: scikit-robot Framework

scikit-robot is designed as a integrated Pure Python framework that systematically addresses each identified research gap. The framework integrates heterogeneous components as a programmable foundation for system integration, primarily targeting modular robot system integration.

A. Knowledge Transfer from EusLisp to scikit-robot

Building upon the rich heritage of EusLisp robotics research, scikit-robot inherits core architectural principles while adapting them for modern Python-based workflows and reconfigurable robotics requirements.

B. Integrated Toolchain for Modular URDF Manipulation

scikit-robot addresses the "decomposition-composition-execution" gap by providing an integrated URDF toolchain that connects decomposition to composition.

modularize-urdf: Decomposes monolithic URDFs into reusable, parameterized xacro macros.

change-urdf-root: Reconfigures (composes) URDF hierarchical structures by making any link the new root through programmatic API calls. This enables modular connections at connection points such as arm tips or body sides.

convert-urdf-mesh: Optimizes 3D meshes to improve simulation execution efficiency.

visualize-urdf: Immediately displays generated and transformed URDFs in 3D, enabling interactive verification of correct kinematic structures through visualization of model placement positions and joint positions.

This toolchain enables reconfiguration workflows that previously required manual work and multiple software tools to be completed within a single Python script.

C. Hash-based Model Management and Distribution

scikit-robot addresses runtime update challenges by operating in a Pure Python environment independent of ROS constraints, providing its own state management mechanisms.

Solving URDF Model Ambiguity Problems: Computes hash values including not only URDF file bodies but also binary data of all referenced meshes and textures. This ensures model identity of reconfigured models and resolves misidentification problems seen in conventional approaches.

Automatic Model Distribution (hash-based URI scheme): Uses hash values as unique identifiers to transparently download and load models and all assets from local caches or remote servers. This realizes easy model management like pip install and provides a robust foundation for programmatically switching between different configuration models at runtime. This proposes an alternative for dynamic model management that replaces ROS's immutable parameter server.

D. Enhanced Visualization for Limited URDF Semantics

Rather than attempting to fundamentally extend URDF's semantic expressiveness—which would require changes to the standard itself—scikit-robot addresses the practical implications of this limitation through comprehensive visual feedback and intuitive interaction paradigms.

The framework provides immediate visual feedback for every model transformation, enabling developers to understand not just the mathematical correctness of transformations but also their physical implications and constraints. This approach acknowledges URDF's semantic poverty while providing practical solutions that allow developers to work effectively within these constraints. The enhanced visual semantics serve as a compensatory mechanism, enabling developers to intuitively comprehend complex reconfiguration operations and validate design decisions through immediate graphical feedback rather than abstract parameter interpretation.

E. System Architecture Overview

scikit-robot is an integrated framework built around three core principles: modularity, efficiency, and accessibility. In the system architecture, users write Python code while various tools (modularize-urdf, change-urdf-root, convert-urdf-mesh, visualize-urdf) collaborate to generate URDFs. Behind the scenes, CascadedCoords handles coordinate management while the hash system guarantees model integrity. Additionally, the FormaMotus Blender plugin enables 3D visualization and GUI operations of generated URDF files. Ultimately, this provides an integrated workflow usable across simulators (PyBullet, etc.), real robots, and visualization systems. The main components are: (1) unified coordinate management via CascadedCoords, (2) comprehensive URDF toolchain, (3) hash-based model management system, (4) high accessibility through Pure Python implementation, and (5) FormaMotus integrated visualization platform.

F. Core Functional Architecture of scikit-robot

scikit-robot consists of efficient coordinate transformation classes and robot model class groups including inverse kinematics computation, automatically generating robot models from URDF and providing integrated 3D visualization, ROS interfaces, and motion planning.

G. CascadedCoords: Unified Coordinate Management

Unlike conventional approaches that recalculate transformations from zero, CascadedCoords provides efficient hierarchical coordinate transformations. The tree structure realizes: incremental updates (recalculating only affected branches during configuration changes), lazy evaluation (computing only when necessary), and memory efficiency (shared coordinate objects).

H. Detailed Components of the URDF Toolchain

modularize-urdf: From Static to Modular Converts monolithic URDF files into reusable xacro macros. The conversion process includes: (1) namespace management (preventing name collisions through prefix addition), (2) parameter extraction (identifying configurable parameters), (3) connection point definition (specifying connector links), and (4) macro generation (creating appropriately parameterized xacro templates).

change-urdf-root: Solving Connection Problems Implements a root change algorithm that addresses mismatches between module design and connection requirements through three steps: (1) *Path Discovery*: Uses depth-first search (DFS) to find the kinematic chain between old and new roots, (2) *Coordinate Inversion*: Systematically inverts transformation matrices along the discovered path using joint reversal algorithms that preserve physical constraints, and (3) *Topology Reconstruction*: Rebuilds the URDF XML tree structure with new parent-child relationships while maintaining joint limits and dynamic properties. The algorithm ensures kinematic equivalence by preserving forward kinematics solutions while enabling flexible connection topologies. This generates physically equivalent but topologically different URDFs, enabling connection patterns that are difficult to achieve with standard tools.

convert-urdf-mesh: Mesh Optimization Optimizes 3D meshes while preserving visual quality: texture-preserving decimation, format conversion (STL, OBJ, DAE, PLY, 3DXML), batch processing, and quality control (configurable optimization levels). Figure 2 shows the URDF generation workflow from CAD through scikit-robot.

Robot design begins with CAD software such as Fusion 360, SolidWorks, or Onshape. The CAD model is then

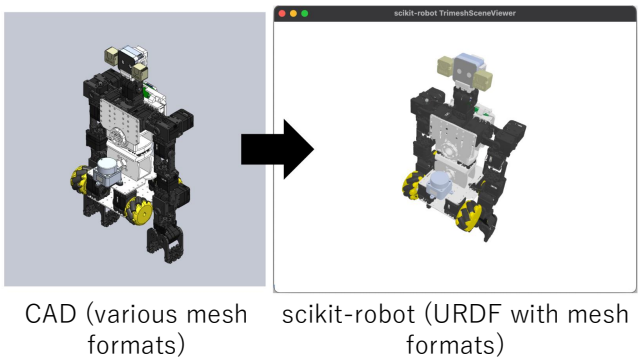

Fig. 2. Robot model development workflow from CAD through scikit-robot

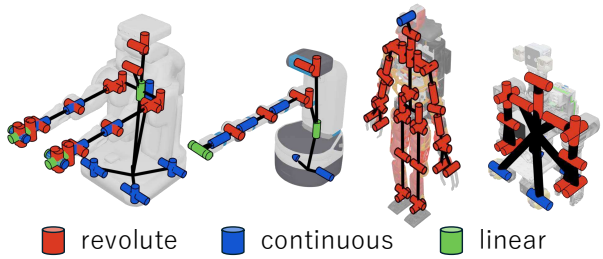

Fig. 3. Joint Structure Diagram of robot displayed in FormaMotus Blender plugin

exported to URDF format using tools like Onshape-to-robot or SolidWorks-to-URDF exporter. scikit-robot processes this URDF through its integrated toolchain: modularize-urdf converts monolithic models to modular xacro macros, change-urdf-root enables connection at any link, and convert-urdf-mesh optimizes 3D assets. The resulting optimized URDF can then be used across simulators, real robot systems, and high-quality visualization environments. FormaMotus Blender plugin[1] provides Joint Structure Diagram visualization as shown in Figure 3, enabling intuitive understanding of complex robot topologies.

I. Hash-based Model Management System

To solve the misidentification problems of conventional metadata-based duplicate detection, a comprehensive URDF hash system was introduced.

Comprehensive Content Hashing Rather than simple URDF hashing, generates comprehensive hashes including: (1) URDF file body (XML structure and all attribute values), (2) mesh files (all referenced 3D meshes), and (3) texture files (mesh-related texture images). The system uses SHA-256 cryptographic hash function for generating content-based identifiers and W3C XML Canonicalization (C14N) standard [24] for XML normalization.

Hash-based URI Scheme and Automatic Distribution Uses hash values as unique model identifiers, enabling automatic downloading and local caching of models and all associated assets. This infrastructure supports distributed development workflows where team members can seamlessly share and access model variations without manual file management.

[1]FormaMotus uses scikit-robot as its backend and is available as open-source software: https://github.com/iory/formamotus

IV. Experimental Validation

A. Resolving Model Ambiguity with Comprehensive Hashing

1) Experimental Setup: To validate the effectiveness of our comprehensive hashing approach, we conducted experiments using a dataset of 320 URDF files [11] collected from various sources including academic research, industrial applications, and open-source repositories.

2) Hash-based Accuracy Analysis: Our comprehensive hashing approach successfully identified and resolved cases of model ambiguity that were missed by conventional metadata-based approaches. The validation demonstrates that content-based hashing provides significantly more accurate duplicate detection compared to filename-based or metadata-based methods, which often fail to distinguish between substantially different robot models that share similar names or metadata.

B. Performance Gains from Mesh Caching in Modular Robots

1) Experimental Target: Modular Robot Dataset: We evaluated performance using datasets from modular robots including tycoon modular robot system [25] and griphis manipulator [26] various configurations of multi-degree-of-freedom modular robots with shared mesh assets between modules.

Figure 4 demonstrates the practical application of our proposed methodology to real modular robot systems. The left side shows individual robot modules with their original root-links (indicated by red-blue-green coordinate axes), while the right side shows the successfully connected composite robots. For the Griphis manipulator, the original root-link is positioned at the base rather than at the connection point (gripper tip), which prevents natural modular connections. Our dynamic root transformation algorithm enables repositioning the root-link to the connection point, facilitating module integration and creating physically meaningful composite robot models.

Such composite robot models combining multiple individual modules are rarely shared within the robotics community, despite their significant value for advancing reconfigurable robotics research. By providing these integrated models and the tools to create them, scikit-robot contributes to building a foundation for more widespread development and validation of reconfigurable robotic systems. This addresses a critical gap in available datasets and modeling resources for the reconfigurable robotics community.

2) Performance Comparison Results: scikit-robot demonstrates performance advantages in scenarios with high module reuse, achieving up to 5.1× acceleration with the Tycoon modular robot compared to conventional approaches through mesh caching and reuse strategies.

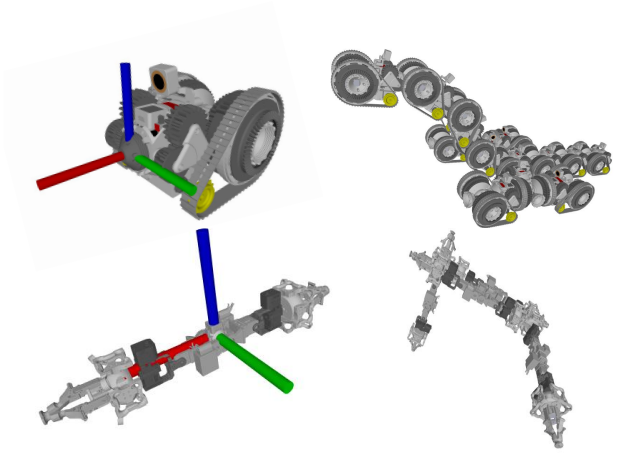

Fig. 4. Root-link transformation enabling modular robot composition: individual modules (left) with original root-links shown as red-green-blue coordinate systems, and connected composite robots (right) for Griphis and Tycoon systems. The dynamic root transformation algorithm repositions root-links to connection points, enabling natural modular assembly

TABLE II
MODULAR ROBOT URDF LOADING PERFORMANCE COMPARISON

Robot System	yourdfpy	scikit-robot	Speedup
Griphis manipulator	1.8s	0.6s	3.0×
Tycoon modular robot	8.7s	1.7s	5.1×

3) Bottleneck Analysis through Validation: Performance bottlenecks in conventional approaches primarily stem from redundant mesh loading and processing. Our caching strategy eliminates these redundancies while maintaining full model fidelity.

4) Modular Robot-Specific Challenges: Modular robots present unique challenges due to high asset reuse ratios and complex dependency graphs. Our hash-based approach handles these challenges by treating shared assets as first-class entities in the model management system.

C. Overall Performance and Reliability Benchmark

1) Validation Target Dataset: We conducted comprehensive benchmarking using diverse URDF datasets [11] including industrial robots, research platforms, and simulated environments to evaluate both performance and reliability across different scenarios.

2) Performance and Reliability Comparison Results: scikit-robot outperforms existing solutions across multiple metrics while maintaining compatibility with standard URDF specifications.

TABLE III
COMPREHENSIVE PERFORMANCE COMPARISON: YOURDFPY VS SCIKIT-ROBOT

Metric	yourdfpy	scikit-robot	Improvement
Loading Speed	1.69s	0.87s	1.94×
Error Rate	3.2%	0%	100% reduction

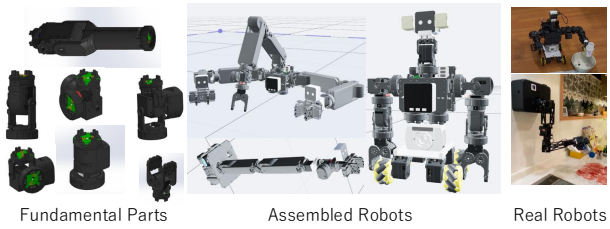

Fig. 5. Modular assembly capabilities: fundamental robot components (left) [27], various assembled configurations (middle), and real robot deployment (right) demonstrating complete design-to-deployment workflow

3) Detailed Analysis: The performance improvements stem from our integrated approach to asset management, efficient coordinate transformations, and intelligent caching strategies. Zero error rate demonstrates the robustness of our comprehensive validation and testing framework.

D. Qualitative Assessment of the End-to-End Workflow

Beyond quantitative performance metrics, we conducted qualitative validation to demonstrate the practical applicability of scikit-robot for real-world reconfigurable robotics development. Figure 5 illustrates the comprehensive modularity enabled by our framework.

The left side shows fundamental robot components: individual actuator modules, sensor units, structural connectors, and specialized end-effectors. These standardized parts serve as building blocks for creating diverse robot configurations. The right side demonstrates various assembled robot configurations achievable through different combinations of these fundamental components, including serial manipulators, parallel mechanisms, and hybrid structures.

This modular approach provides several critical advantages: (1) **Rapid Prototyping**: Engineers can quickly assemble and test new robot configurations without custom hardware development, (2) **Design Validation**: Physical feasibility and kinematic constraints can be verified before actual hardware construction, (3) **Immediate Deployment**: Once validated in simulation, the exact same model definitions can be seamlessly transferred to control real robot hardware, and (4) **Systematic Design Space Exploration**: The standardized component library enables systematic investigation of optimal configurations for specific applications.

scikit-robot's integrated toolchain ensures that transitions from conceptual design through simulation validation to real robot deployment occur without model translation errors or compatibility issues. This workflow reduces the traditional barriers between design iteration and physical implementation in reconfigurable robotics. The rightmost section of Figure 5 demonstrates successful real-world deployment, showing the assembled modular robots operating in physical environments, thereby validating the complete design-to-deployment pipeline enabled by our framework.

V. DISCUSSION

The experimental results demonstrate that scikit-robot addresses the key challenges in reconfigurable robotics modeling. The framework's integrated approach provides both

immediate practical benefits and establishes a foundation for future research in adaptive robotic systems.

To ensure its accessibility and long-term viability, scikit-robot is publicly available as an open-source project on GitHub (https://github.com/iory/scikit-robot) and is distributed as a standard Python library. Continuous integration with daily automated testing ensures the framework's reliability. This commitment to open-source development not only provides a stable tool for the community but also facilitates that scikit-robot will be actively maintained and enhanced with features specifically targeting the needs of modular and reconfigurable robotics.

Looking ahead, a key future direction is to develop novel user interfaces that fully leverage the dynamic nature of reconfigurable robots. We plan to create an interface that overlays virtual robot models onto their real-world counterparts, allowing users to intuitively design and trigger automatic changes to the robot's physical configuration. Such a system would bridge the gap between digital design and real-world execution, truly unlocking the core advantage of reconfigurable systems.

VI. Conclusion

We addressed three critical barriers preventing widespread adoption of reconfigurable robotics: the inability to seamlessly transition between model decomposition and composition during execution, the absence of mechanisms for dynamic kinematic root modification, and the insufficient expressiveness of current modeling standards. Our solution, scikit-robot, provides a unified framework that addresses these challenges for reconfigurable robotics development.

Our framework establishes a design-to-deployment workflow through three key points: (1) an integrated toolchain enabling model decomposition and composition, (2) root transformation algorithms that reduce physical connection constraints, and (3) a hash-based management system supporting simulation-to-hardware integration. We demonstrate the effectiveness through validated deployment on multiple physical modular robot systems.

scikit-robot addresses the gap between research and implementation, providing a practical foundation for reconfigurable robotics. By reducing the barriers between simulation-based design and real-world deployment, our framework enables researchers and developers to focus on creating adaptive and intelligent robotic systems. This work represents a move from fragmented development processes to unified workflows that support the development of reconfigurable robotics applications.

References

[1] E. Klavins, R. Ghrist, and D. Lipsky, "A grammatical approach to self-organizing robotic systems," *IEEE Transactions on Automatic Control*, vol. 51, no. 6, pp. 949–962, 2006.

[2] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems: Challenges and opportunities for the future," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[3] T. Fukuda and S. Nakagawa, "Dynamically reconfigurable robotic system," in *IEEE International Conference on Robotics and Automation*, 1988, pp. 1581–1586.

[4] M. Yim, D. G. Duff, and K. D. Roufas, "Polybot: a modular reconfigurable robot," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 514–520.

[5] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-tran: Self-reconfigurable modular robotic system," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 431–441, 2002.

[6] S. Hauser, M. Mutlu, P.-A. Léziart, H. Khodr, A. Bernardino, and A. J. Ijspeert, "Roombots extended: Challenges in the next generation of self-reconfigurable modular robots and their application in adaptive and assistive furniture," *Robotics and Autonomous Systems*, vol. 127, p. 103467, 2020.

[7] C. Liu, Q. Lin, H. Kim, and M. Yim, "Smores-ep, a modular robot with parallel self-assembly," *Autonomous Robots*, vol. 47, pp. 211–228, 2023.

[8] L. Rossini, E. Romiti, A. Laurenzi, F. Ruscelli, M. Ruzzon, L. Covizzi, L. Baccelliere, S. Carrozzo, M. Terzer, M. Magri *et al.*, "Concert: a modular reconfigurable robot for construction," *arXiv preprint arXiv:2504.04998*, 2025.

[9] ROS Wiki, "Urdf," http://wiki.ros.org/urdf, 2025, [Online; accessed 08-15-2025].

[10] D. Tola and P. Corke, "Understanding urdf: A survey based on user experience," in *International Conference on Automation Science and Engineering*, 2023, pp. 1–7.

[11] ——, "Understanding urdf: A dataset and analysis," *IEEE Robotics and Automation Letters*, vol. 9, no. 5, pp. 4479–4486, 2024.

[12] Open Source Robotics Foundation, *SDFormat*, http://sdformat.org/, 2025, [Online; accessed 08-15-2025].

[13] ROS Community, *Xacro*, https://github.com/ros/xacro/wiki, 2025, [Online; accessed 08-15-2025].

[14] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[15] C. Eppner, "yourdfpy: Modern python urdf parser and manipulator," https://github.com/clemense/yourdfpy, 2025, [Online; accessed 08-15-2025].

[16] ROS Community, "urdf_parser_py: Python urdf parsing library," https://github.com/ros/urdf_parser_py, 2025, [Online; accessed 08-15-2025].

[17] M. Matl, "urdfpy," https://github.com/mmatl/urdfpy, 2025, [Online; accessed 08-15-2025].

[18] M. Chignoli, J.-J. Slotine, P. M. Wensing, and S. Kim, "Urdf+: An enhanced urdf for robots with kinematic loops," in *International Conference on Humanoid Robots*, 2024, pp. 197–204.

[19] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integration*, 2019, pp. 614–619.

[20] T. Matsui and M. Inaba, "Euslisp: an object-based implementation of lisp," *Journal of Information Processing*, vol. 13, no. 3, pp. 327–338, 1991.

[21] J. Külz, M. Mayer, and M. Althoff, "Timor python: A toolbox for industrial modular robotics," in *International Conference on Intelligent Robots and Systems*, 2023, pp. 424–431.

[22] ADVR-Humanoids, "Tool for urdf generation of modular robots," https://github.com/ADVRHumanoids/modular_hhcm, 2025, [Online; accessed 08-15-2025].

[23] R. Team, "Onshape to robot," https://github.com/Rhoban/onshape-to-robot, 2025, [Online; accessed 08-15-2025].

[24] World Wide Web Consortium, "Canonical xml version 1.1," https://www.w3.org/TR/2008/REC-xml-c14n11-20080502/, 2008, [Online; accessed 08-15-2025].

[25] K. Matsuo, T. Makabe, I. Yanokura, K. Kojima, K. Okada, and M. Inaba, "Configuration of modular robot that combines torque transmission and joint drive functions by complementary intermittent gear mechanisms and serial manipulator realization," in *JSME Conference on Robotics and Mechatronics*, 2024, pp. 1A1–M03.

[26] S. Morita, K. Matsuo, T. Makabe, I. Yanokura, K. Kojima, and K. Okada, "Development of multi-joint modular robot capable of grasping objects in both single and combined states by using docking-gripping mechanism," in *JSME Conference on Robotics and Mechatronics*, 2025, pp. 1P1–E11.

[27] T. Makabe, N. Yamaguchi, I. Yanokura, and K. Okada, "Development and application of 3d-printable screw electromechanical coupling," in *JSME Conference on Robotics and Mechatronics*, 2025, pp. 1P1–E09.