

GCP-HOLO: Generating High-Order Linkage Graphs for Path Synthesis

Mitchell B. Fogelson

Mem. ASME

Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: mfogelson@andrew.cmu.edu

Conrad Tucker

Mem. ASME

Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: conradt@andrew.cmu.edu

Jonathan Cagan¹

Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: cagan@cmu.edu

One degrees-of-freedom (1DOF) linkages are persistent in mechanical systems. However, designing linkages to follow a desired path, known as path synthesis, is challenging due to non-linearities, combinatorial nature, and strict geometric constraints. Current state-of-the-art algorithms cannot scale well to linkages with higher-order linkage graphs, which are required to satisfy more complicated paths for new mechanical systems, such as hopping and flying robots. One reason for this is that state-of-the-art algorithms spend the majority of the time exploring constraint-violating designs. This work uses an Assur group 0DOF linkage as a graph grammar rule to modify both linkage graph and spatial parameters, ensuring all designs are valid 1DOF linkages. Using this graph grammar, this paper formulates linkage path synthesis as a tree search and uses a deep reinforcement learning (DRL) agent to search the space of kinematically feasible planar 1DOF linkages. This paper introduces a method using a graph convolution policy for high-order linkage graph optimization (GCP-HOLO). An anytime algorithm, GCP-HOLO outputs linkages with 1–8 loops (4–16 bars) efficiently. When comparing the GCP-HOLO formulation to a recent state-of-the-art paper that solves a mixed integer conic program, GCP-HOLO generates sets of solutions of varying linkage complexities to eight test trajectories in a quarter of the time. Extending GCP-HOLO with a global node optimization, such as covariance matrix adaptation evolutionary strategy, the results quickly converge to finding better solutions for 4/8 tests, with the whole pipeline capable of a 13X speed increase.
[DOI: 10.1115/1.4062147]

Keywords: design automation, machine learning, mechanism synthesis

1 Introduction

In the long history of the path synthesis problem, there still lacks a scalable method for simultaneous graph and geometric parameter design for path synthesis of high-order linkage graphs. This paper defines a high-order linkage graph as having three or more loops or greater or equal to an eight-bar linkage. A solution for this problem can help engineers explore new applications for linkages in designing mechanical systems. Linkages consistently have resurgences in the design of mechanical systems due to their ability to transform circular motion into any arbitrary path and improve the mechanical advantage of the end effector, such as the force and torque advantages. Recently, with stronger computational tools [1], the design of more complex linkages advanced areas in modern robot designs [2–4]. The eight-bar linkage design reduced the number of actuators used on the robot enabling highly dynamic behavior with overall lower weight and cost [2,4]. Current systems still primarily focus on pure geometric optimization, which requires expert knowledge from the end-user, such as which linkage graphs will lend themselves to the desired trajectory. This work seeks to reduce the expertise needed of the end-user in solving path synthesis by introducing a method called graph convolution policy for high-order linkage graph optimization (GCP-HOLO)² which:

- (1) explores both linkage graph and linkage geometry simultaneously;
- (2) provides sets of solutions with varying linkage graphs topologies rather than a point solution;

- (3) can be augmented with other design constraints specific to the desired task.

The problem of linkage design is challenging due to the mixed discrete and continuous design variables, non-linearities, combinatorial nature, and strict geometric constraints. This paper focuses on linkages defined by rigid bars connected by revolute joints. The configuration of bars and joints defines the linkage graph, and the location of the revolute joints or bar lengths defines the linkage geometry. The linkage graph configuration problem is a combinatorial discrete number problem with a long history of research related to the generation of unique configurations [5]. A notable solution to this problem was by Tuttle, who found all possible 1DOF mechanisms with up to 1–6 loops (4–12 bars) to be 4.3×10^6 configurations [6]. The linkage geometry problem is a continuous problem and has an equally long history of investigation. A pivotal contribution in this area was from Freudenstein, who created an analytical approach to solving the path synthesis problem for a one-loop (four-bar) linkage [7]. It may be surprising that the progress in solving the path synthesis problem for higher-order linkage graphs is still very much unsolved. Recent works like Ref. [8] found the 1.5×10^6 solutions to a two-loop (six-bar) linkage graph for a path synthesis problem. This paper considers three-loop (eight-bar) or greater linkages as high-order linkage graphs because there are no analytical approaches to solving the path synthesis problem. A few works, such as Refs. [9–11], have attempted to solve the path synthesis problem by combining the linkage graph and geometry. These methods greatly support the end-user, who may not be an expert in mechanism design. The linkage designs for all these problems must satisfy the 1DOF constraint and the strict physical constraints of the rigid bar connections defined by the revolute joints.

This paper provides an automated method to address the three goals defined prior. This paper presents a fixed homogeneous action space to iteratively grow a base linkage using the Assur

¹Corresponding author.

²https://github.com/mfogelson/gcp_holo

Contributed by the Mechanisms and Robotics Committee of ASME for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received July 4, 2022; final manuscript received March 8, 2023; published online April 19, 2023. Assoc. Editor: Alba Perez Gracia.

group RRR 0DOF kinematic chain [12]. The problem is formulated as a tree search where a reinforcement learning (RL) agent is trained to learn a search heuristic to efficiently find designs that minimize the coupler trajectory to the goal. The search agent explores various depths of the graph to consider different linkage topologies to present as a solution set.

The paper compares GCP-HOLO to a recent work that attempts to solve a similar problem using a mixed integer conic program (MICO) formulation. The paper shows an example of an application of a linkage system with secondary design constraints by applying the method to designing a linkage for a walking system based on the TrotBot footpath. Finally, a deeper evaluation of reinforcement learning variants' performance is considered. This test compares the GCP-HOLO with on-policy updates, GCP-HOLO with off-policy, and a baseline random search. The main contributions of this work are as follows:

- (1) A tree search formulation of linkage path synthesis problem with RL search agent;
- (2) Linkage latent state representation using graph convolution network (GCN);
- (3) Fast linkage generation and simulation using 0DOF Assur group rule and symbolic kinematics;
- (4) An Open AI Gym environment for linkage design that can be used as a testbed for future RL algorithms.

2 Related Works

2.1 Fixed Linkage Graph Optimization. Many approaches to the path synthesis problem fix the discrete linkage graph a priori and optimize only the continuous geometric parameters to minimize the distance to the desired coupler trajectory. A state-of-the-art method for path synthesis with high precision uses a dimensional synthesis method known as homotopy continuation [1]. That method converts the non-linear system of equations that define the linkage in the various key point positions into a high-order polynomial. The formulation is then converted into a root-finding problem which is solved for the geometric parameters. That method can solve a path that consists of nine sample points for a four-bar linkage [13] and up to 11 sample points for a six-bar linkage [14]. The maximum number of sample points is determined by the number of unknown parameters defining the mechanism, for example, link lengths or joint positions. A key limitation of that method is the computational complexity and restriction on the number of precision points. Because of these scaling issues, no further extensions for higher-order linkages exist. That work solved the exact solutions that satisfy the precision points for low-order linkage graphs (four-bar and six-bar), where the problem this paper addresses is path synthesis for high-order linkage graphs (\geq eight-bar).

Stochastic optimization techniques, such as the covariance matrix adaptation evolutionary strategy (CMA-ES), are an approach that is capable of optimizing geometric parameters of higher-order linkages [15]. That method converges to reasonable solutions when the design parameters are bounded and strong initial solutions are provided. That method can be computationally expensive as most designs the method considers are kinematically invalid due to the strict geometric constraints. CMA-ES also fails when the optimization is unbounded, as many solutions tend towards infinity or propose unrealistic design parameters. GCP-HOLO can be extended using CMA-ES to improve the solutions through small changes to the joint locations. Section 4.1 shows an example of GCP-HOLO's solutions improved with CMA-ES.

The linkage kinematics can be described as closed-form equations for a class of linkages proposed by Ref. [16], known as simple kinematic loops. Bächer et al. introduced a gradient-based method to update the continuous parameters in a real-time interactive system [16]. That work showed that gradient descent is well suited when exploring the spatial parameters of a single linkage graph where a designer can influence different aspects of the linkage, such as scaling, specific node trajectories, or path constraints. That gradient-based approach quickly settles in a local

minimum and only provides point-based solutions. That method could also be used to improve the final designs of GCP-HOLO, but since it natively does not consider the linkage graph variants, that work is not considered further in this paper. These fixed linkage graph approaches to path synthesis designs cannot simply be extended by iterating over all linkage graphs as the number of variations grows exponentially, with over 4.5×10^6 linkage graphs for linkages with less than 6 loops or 14 bars [6].

2.2 Simultaneous Linkage Design Optimization. Recent work by Pan et al. [11] represents the problem as a MICO and solves for the optimal solution using a branch and bound method. That work showed significant progress in solving the planar path synthesis where both linkage graph and geometric parameters are solved simultaneously. However, some shortcomings are that the method cannot guarantee that intermediate designs are feasible, which requires the method to converge before a solution can be considered. Another limitation is that the computational complexity grows exponentially as the approach considers designs with higher linkage graphs, more sample points, or finer discretization. Finally, the method only provides a point solution, which may not be preferred in design automation as there are many considerations in the design that might not be expressed in the objective function, such as manufacturability and assembly. In this work, GCP-HOLO is an anytime algorithm that generates only kinematically feasible linkages addressing the intermediate design feasibility. GCP-HOLO also suffers from computational complexity but to a much lesser extent since invalid designs are prohibited from the formulation of the grammar rule. Finally, GCP-HOLO outputs a set of linkage designs of varying linkage topologies rather than a point solution. While MICO has some optimality guarantees, the planar path synthesis problem contains many local minima with objectives close to the optimal solution, as shown in the results in this paper. Therefore, this work can find several strong solutions in significantly less time, lending itself advantageously in certain situations. That work is compared to the current method in Sec. 4.1 as it most similarly addresses a comparable problem.

Zhao et al. [17] provided a mixed exact and approximate motion realization method that finds the optimal planar dyad chain. The planar dyad chain can then be converted into four-bar linkage solutions. Their approach converts the desired trajectory through kinematic mapping to a higher dimensional representation, the Image Space, that enables them to solve a least square fitting problem. The solution parameters from the least squares fitting are correlated to the different planar dyad types (RR, PR, RP). The advantage of this approach is that the solution found can have up to four precision poses and an arbitrary number of inexact poses. One limitation of this approach is that the desired trajectory must be satisfied by a planar dyad, which could be hard to determine a priori if the dyad is well suited for the path.

Lipson proposed a genetic algorithm for linkage synthesis that simultaneously acts on the design's linkage graph and spatial parameters with graph grammars [9]. Graph grammars are a set of pre-specified rules that modifies the current state by either adding to the graph, removing parts of the graph, or augmenting the graph structure. The graph grammars that Lipson defined were applied to the rigid bars of the linkage, which requires the forward kinematics (FK) of the new design to be fully re-evaluated. Since the FK must be evaluated thousands of times in genetic algorithms and RL, an efficient method for computing the FK must be considered. This paper defines the grammar rule on the nodes to address this issue, which only requires the added component's FK to be evaluated using the symbolic kinematics method. Another limitation of Lipson's grammar rules is the lack of constraints on the new joint's initial location, which can lead to the evaluation of many kinematically degenerate designs. This paper uses a simplified constraint satisfaction criteria to evaluate the validity of new joint locations quickly. Finally, this paper extends the applications shown in Lipson, which only explored linkage design for a straight-

line synthesis problem. This paper is applied to the more general path synthesis problem.

Vermeer et al. [10] trained a deep RL agent to learn a policy to apply Lipson's grammar rules. To use Lipson's grammar in their RL framework, Vermeer et al. introduced a spatial constraint such that the new node added forms an isosceles triangle. This constraint converts the graph grammar to a fixed action space more suitable for RL. However, that constraint impaired their method's performance in generating designs and failed to achieve comparable results to Lipson's method. That constraint also fails to guarantee a valid design since the isosceles triangle only satisfies the linkage constraints in the first time-step and not the whole trajectory. To address these limitations, this work replaces their spatial constraint with a set of scaffold nodes—a set of potential initial joint locations. This achieves a suitable action space for the RL agent with relaxed design limitations. This paper also uses a simplified constraint satisfaction criteria that check the validity of the node positions through the whole trajectory and not the initial time-step to prevent the evaluation of constraint-violating designs. Vermeer et al. defined the state of the linkage using hand-crafted features. This is inferior to learned feature representation using deep learning [18]. Therefore, in this work, a graph embedding network is used to learn the latent features of the current design state.

GCP-HOLO improves and differs from the prior works by defining a graph grammar rule that does not affect prior revolute joints' trajectories and therefore avoids re-evaluation of the full forward kinematics. GCP-HOLO introduces scaffold nodes to extend the linkage configuration with a simplified constraint satisfaction criteria that quickly identifies the set of valid states, ensuring an anytime algorithm that generates valid 1DOF linkages. GCP-HOLO converts the graph representation of the linkage into a vector through a learned mapping. Finally, GCP-HOLO trains an RL agent to learn a search direction that improves the generation of linkages to satisfy the path synthesis problem.

2.3 Reinforcement Learning in Design. GCP-HOLO was inspired by several recent works that used deep RL agents to find solutions for design automation problems. Similar methods to GCP-HOLO have been used in solving problems related to truss design [19], molecules design [20], and robot design [21,22]. RL-based methods provide a sequential design approach that Raina et al. [19] stated is comparable to human engineering design. You et al. argued that deep RL methods are more robust than other machine learning-based generative methods for designing physical systems, as RL can ensure that all physical constraints of the design problem are satisfied [20]. GCP-HOLO uses reinforcement learning rather than other tree search methods, as it has been shown in [20–23] to outperform search algorithms, such as Monte Carlo Tree Search and best-first search.

Several areas differentiate the RL pipelines in each of those works. First, the state representation of the design can vary. Trusses, molecules, and robot configurations are all fixed systems, whereas linkages are time-varying periodic systems. For example, in Raina et al. [19], images of trusses are used to evaluate the current state of the design. In Refs. [20–22], the design state of the molecule and robots are represented as a graph. This work uses a graph representation to represent the time-varying linkage. Second is the agent design itself; Whitman et al. [21], and Zhao et al. [22], proposed deep Q-network (DQN), an off-policy network, and You et al. [20], used proximal policy optimization (PPO), an on-policy algorithm. An on-policy RL algorithm updates the policy network directly responsible for selecting the action with the most recent data generated from the rollouts. An off-policy RL algorithm keeps track of a history of data generated from the rollouts and randomly samples from this buffer to update the network. This work explicitly compares the different policies' performance for the linkage generation problem.

3 Method

Section 3.1 describes the class of linkages that this work considers and the simulation method to obtain the coupler trajectory. Section 3.2 describes the GCP-HOLO algorithm, which includes Sec. 3.2.1, describing the linkage state representation as a graph and the GCN to learn the latent state representation. Section 3.2.2 defines the action space from the graph grammar rule to explore linkage designs. Section 3.2.3 shows the objective of the path synthesis problem and reward design used to train the reinforcement learning agent. Finally, Sec. 3.2.4 describes the different types of reinforcement learning algorithms that can be used and the training process for the method.

3.1 Linkage Formulation. The kinematic chains this work designs can be understood as 1DOF high-order planar linkage graphs with simple kinematic loops. This section provides a breakdown of each of those terms to better inform the reader. One degrees-of-freedom (1DOF) defines the number of input variables needed to define the linkage pose fully. The type of linkages this paper designs consists of a set of moveable and fixed revolute joints connected by a set of rigid links and a motor input. For the 1DOF system knowing the angle of the motor input is sufficient to know the position of the rest of the mechanism. A complete revolution of the motor generates periodic paths for each moveable revolute joint. Gruebler's formula, shown in Eq. (1), can be used to verify the degrees-of-freedom of the system:

$$\text{DOF} = 3(n - 1) - 2j \quad (1)$$

where n is the total number of links, j is the total number of revolute joints.

In planar mechanisms, all the revolute joints and rigid bars exist in the XY -plane. This is useful for designing linkages in simulation and determining the trajectories of the revolute joints. This can cause conflicts when converting designs from simulation to reality as the manufacturability and linkage assembly aren't considered.

The configuration of the revolute joints and rigid bar components describes the linkage graph. The rigid bars in the linkage create closed loop structures. The linkage community, rather than describing the systems by the loops refers to them as four-bar (one loop), six-bar (two loops), eight-bar (three loops), etc. This paper defines higher-order linkage graphs with linkages that have ≥ 3 active loops, this is because there are no analytical approaches to solving the path synthesis of linkages with ≥ 3 active loops. The Jansen linkage in Fig. 1 is an example of a high-order linkage graph as it contains three loops [24]. This work uses the minimum cycle basis method to find the number of active loops that comprises the linkage graph [25].

In our work, the forward kinematics must be evaluated hundreds of thousands if not millions of times. A recursive approach for solving the forward kinematics for closed loop systems provides faster evaluations than constraint-based methods as described in Ref. [26]. Bächer et al. [16] described a recursive method, symbolic kinematics, to quickly evaluate the forward kinematics for a subset of linkage graphs made up of revolute joints. This method does not work for all linkage graphs but does span 7 out of the 11 planar 1DOF eight-bar linkage topologies described by Tsai and McCarthy [27]. Symbolic kinematics is also not currently suited for systems with prismatic joints, so prismatic joints are not considered in this work. Symbolic kinematics represents the linkage by its joint positions and distance constraints between connected revolute joints. The crank joint location is determined from the motor input angles, given in Eq. (2).

$$x_{\text{crank}}(t) = R(\lambda\theta(t)) * (x_{\text{crank}}(1) - x_{\text{motor}}) + x_{\text{motor}} \quad \forall t \in [2, T] \quad (2)$$

where $R(*)$ is the 2×2 rotation matrix, λ is the direction of the motor with $\lambda = \begin{cases} 1 & \text{CCW} \\ -1 & \text{CW} \end{cases}$, and $x(t)$ is the planar position of the joint at

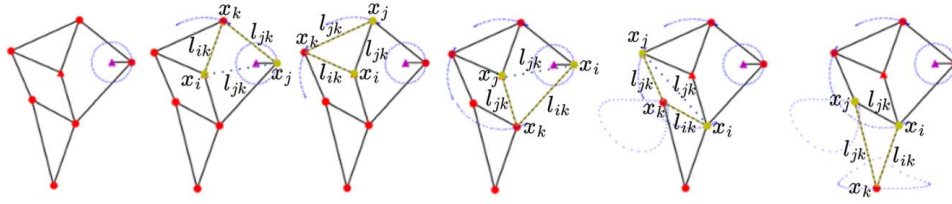


Fig. 1 Symbolic kinematic recursive FK process is shown on Jansen Linkage

time t . This paper evenly spaces the set of discrete angles, $\theta(t)$, from 0 to 2π to generate the linkage trajectories, but any partial set of motor angles can be specified a priori. All other revolute joints, $x_k(t)$, can be found when two neighboring joint locations, $x_i(t)$ and $x_j(t)$, are known. This can be understood geometrically by finding the intersection of two circles centered at the known joints' locations with radii equal to the fixed distances to the unknown joint, respectively. The complete expression for this algebraically can be seen in Eq. (3)

$$x_k(t) = R(\lambda\phi(t)) * \frac{l_{ik}}{l_{ij}(t)}(x_j(t) - x_i(t)) + x_i(t) \forall t \in [2, T] \quad (3)$$

$$\phi(t) = \text{acos}\left(\frac{l_{ij}(t)^2 + l_{ik}^2 - l_{jk}^2}{2l_{ij}(t)l_{ik}}\right)$$

where the fixed side lengths $l_{ik} = \|x_i(1) - x_k(1)\|$ and $l_{jk} = \|x_j(1) - x_k(1)\|$. The final variable side length is $l_{ij}(t) = \|x_i(t) - x_j(t)\|$. From the two solutions, $\lambda = \begin{cases} 1 & \text{Up} \\ -1 & \text{Down} \end{cases}$, the one that matches the given initial joint location, $x_k(1)$, is used and the solution sign is maintained through all time-steps. The reader is referred to Refs. [11,16] for a full description of symbolic kinematics. An example of the iterative processes of symbolic kinematics on the Jansen linkage can be seen in Fig. 1.

3.2 GCP-HOLO. GCP-HOLO is a learning-based directed stochastic search algorithm for high-order linkage graphs to provide sets of solutions to the problem of path synthesis that is amenable to graphic processing unit (GPU) utilization for enhanced performance. A block diagram of the method can be seen in Fig. 2. This section breaks down the GCP-HOLO method in the following way. In Sec. 3.2.1, the state representation of the linkage as well as the GCN is described. In Sec. 3.2.2, the action space, as well as a simplified constraint satisfaction criteria to reduce the number of invalid actions, are explained. In Sec. 3.2.3, the reward design and path synthesis objective are defined. Finally, Sec. 3.2.4 shows the training and evaluation procedures for the reinforcement learning policy.

3.2.1 State Representation. In deviation from the standard graph representation of a linkage, this paper defines a linkage as a graph G , with nodes, \mathbf{V} , representing the revolute joints, and the edges, \mathbf{E} , representing the rigid bars. This deviation is because in the kinematic equations used, the linkage is described by the time-varying position of the revolute joints and distance constraints, rather than fixed reference frames on the rigid links. For the purposes of the graph neural network, the set of edges, \mathbf{E} , are better described using an adjacency matrix \mathbf{A} . The adjacency matrix is a binary symmetric matrix, where $A_{i,j}$ is one if the nodes i and j are connected and otherwise zero. This adjacency representation known as the linkage adjacency matrix in other historical works is most widely used [5]. The numeric labeling of the nodes must

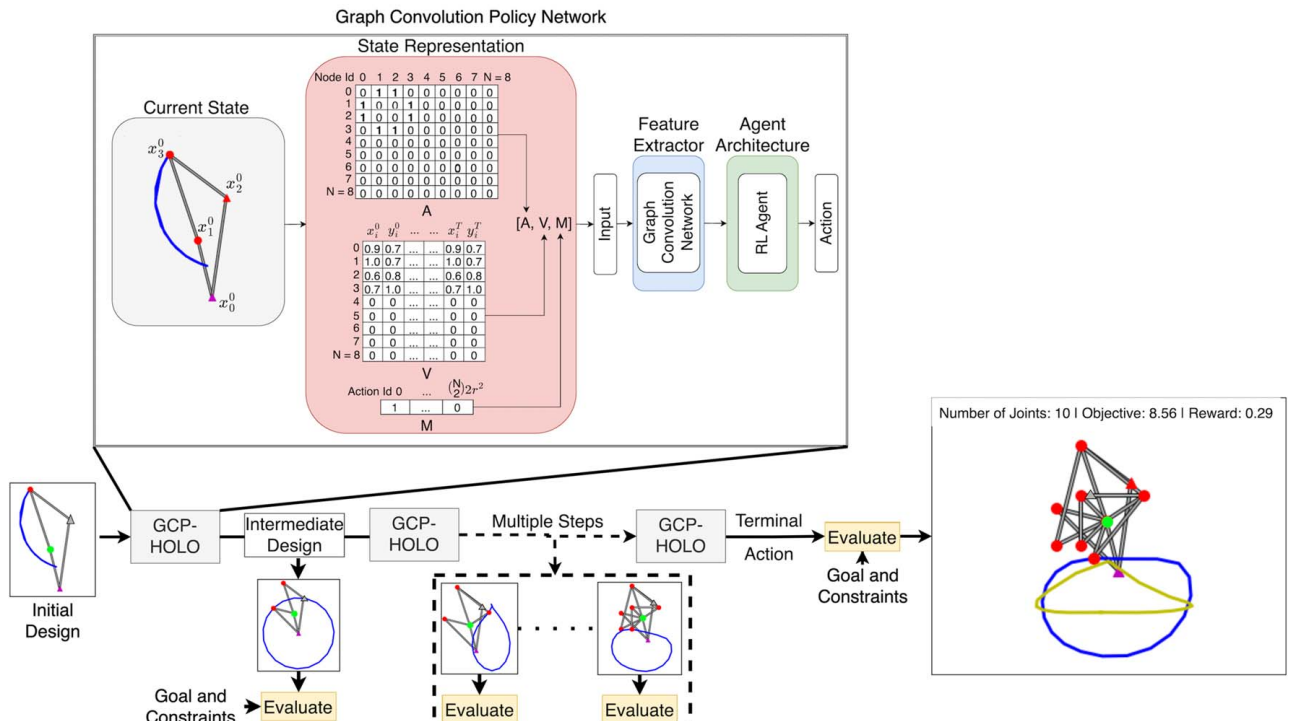


Fig. 2 Block diagram of GCP-HOLO linkage design

follow the hierarchy defined by Ref. [16] such that higher number node indexes are dependent on lower number node indexes. This is always satisfied with the graph grammar rule presented in Fig. 3.

It is typical to map a system's state representation into a vector in an RL pipeline using a feature extractor, for example, converting the linkage graph G to a vector equivalent. Learned feature extractors using neural networks outperform hand-crafted approaches. The feature extractor in GCP-HOLO is a GCN [28] that maps the linkage graph previously described to a lower dimensional vector that encodes pertinent information about the graph. The GCN takes in the node features \mathbf{V} along with the adjacency matrix \mathbf{A} , which it uses as a convolutional filter, to linearly combine the connected node features. The convolution is followed by a fully-connected layer that applies a non-linear activation function to the convolved features. The parameters of the GCN are updated using gradient descent throughout the training process. The GCN this work uses is the same architecture as Zhao et al. [22], shown in Fig. 7—further optimization on the feature extractor network architecture could provide additional performance improvements. Graph networks are permutation invariant and equivariant, meaning that node index permutations do not affect the output of the feature extractor [29]. This is very useful for this problem because sets of actions applied in different orders can lead to the same linkage but different node index ordering. The mapping of these distinct linkage graphs from the graph network will be the same. The feature extractor and the agent are trained end-to-end, improving the learned mapping over time. A diagram of the RL pipeline used in this work can be seen in Fig. 2.

3.2.2 Action Space. The grammar rule in this work is defined such that prior evaluated revolute joint kinematics are unchanged. By avoiding the complete FK evaluation at every step, the speed was 26% faster. The left-hand side of the rule selects two nodes, i and j , from the current linkage and attaches them to a new node k within the design space. The grammar rule can be seen in Fig. 3, where $x_i(t)$ is the position of node i at time $t = 1$.

To avoid constraint-violating positions for the new revolute joint, $x_k(t)$, a simplified constraint satisfaction criteria is presented, shown in Fig. 4. First, a discrete set of positions, known as scaffold nodes, is considered for $x_k(1)$. Next, the fixed link lengths defined by each scaffold node, $l_{ik} = \|x_i(1) - x_k(1)\|$ and $l_{jk} = \|x_j(1) - x_k(1)\|$, along with the potentially varying length defined by the revolute joints, $l_{ij}(t) = \|x_i(t) - x_j(t)\|$, are evaluated for triangle inequality conditions for all time-steps. This check can be evaluated quickly through vector programming and is described in Algorithm 1. While this does not prevent all invalid designs from being able to be considered, it prohibits most of them. In a small test of 1000 random roll-outs, 81% of the designs generated were kinematically invalid without this heuristic, whereas only 2% of designs were invalid when using the heuristic. This allows for the search algorithm to

focus more on finding the coupler trajectory that best matches the desired goal, versus evaluating lots of invalid designs. To further prohibit all constraint-violating designs, such as cases with singularities, a description is provided at the end of Sec. 3.2.3 when discussing the action acceptance criteria.

Algorithm 1 Valid scaffold nodes

Require: x_i, x_j, S	▷ The trajectory of two revolute joints and the grid of scaffold node locations
$l_{ij}(t) \leftarrow \ x_i(t) - x_j(t)\ \forall t \in [1, T]$	▷ The distance between node i, j in a revolution
$l_{ik} \leftarrow \ x_i(1) - S_k\ \forall k \in [1, K]$	▷ The distance between initial position of node i , scaffold node k
$l_{jk} \leftarrow \ x_j(1) - S_k\ \forall k \in [1, K]$	▷ The distance between initial position of node j , scaffold node k
$V_k = \text{all}(l_{ik} + l_{jk} > l_{ij}(t) \forall t \in [1, T]) \wedge$ $\text{all}(l_{ik} + l_{ij}(t) > l_{jk} \forall t \in [1, T]) \wedge$ $\text{all}(l_{ij}(t) + l_{jk} > l_{ik} \forall t \in [1, T]) \forall k \in [1, K]$	
return V	

The total number of actions at any step in the design search is $2 \binom{n}{2} r^2$, where n is the total number of nodes in the current state and r is the resolution of the discretized design space and each action can be either an intermediate or a terminal action. The number of scaffold nodes is set by a user-specified resolution, r , for the x - and y -axis. As the resolution tends toward infinity, $r \rightarrow \infty$, the discrete representation is equivalent to the continuous problem. The higher resolution discretization comes at a computation cost in runtime performance. The set of valid actions for all node pairs, (i, j) , is described in Algorithm 2. The valid actions for each set of node pairs are only computed once and can be cached for all future steps within an episode as the valid actions for that set of nodes do not change as new nodes are added to the graph. Where most prior work spends the majority of time evaluating degenerate designs, Algorithm 1 and Algorithm 2 prohibit the evaluation of most constraint-violating designs.

Algorithm 2 Valid actions

Require: $x_{1:n}$	▷ The trajectory of all revolute joints
$mask \leftarrow \emptyset$	▷ Initialize action mask
$n \leftarrow \text{NumberOfNodes}(x_{1:n})$	▷ Get current number of nodes
$combs \leftarrow \text{combinations}(n, 2)$	▷ Get all i, j node pairs
for $(i, j) \in \text{combs}$ do	
$mask_{i,j,k} \leftarrow \text{ValidScaffoldNodes}(x_i, x_j, S) \forall k \in [1, K]$	▷ Update mask with valid scaffold nodes
end for	
return $mask$	

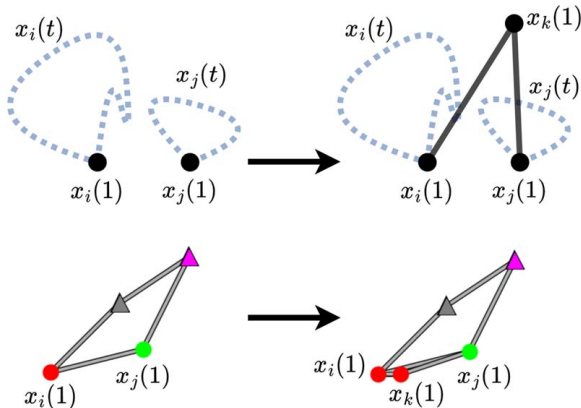


Fig. 3 0DOF Assur group graph grammar rule

When the agent applies a terminal action, the final design can be pruned to only contain the revolute joints which are actively contributing to the coupler trajectory. This is done by recursively traversing from the coupler node to the root, keeping only the parent nodes that were used to solve the FKs for the coupler. All nodes that do not influence the coupler, are unnecessary and do not affect the kinematics of the coupler node. After pruning, the number of active loops that make up the linkage graph can be determined, and the reward of the linkage is determined. Figure 5 shows a visualization of the sequential actions starting with a random valid root node and evaluating with respect to the Jansen footpath and Fig. 6 shows a pruning from a terminal design.

3.2.3 Design Objective. The terminal node added to the linkage is known as the coupler node, x_N , and defines the trajectory

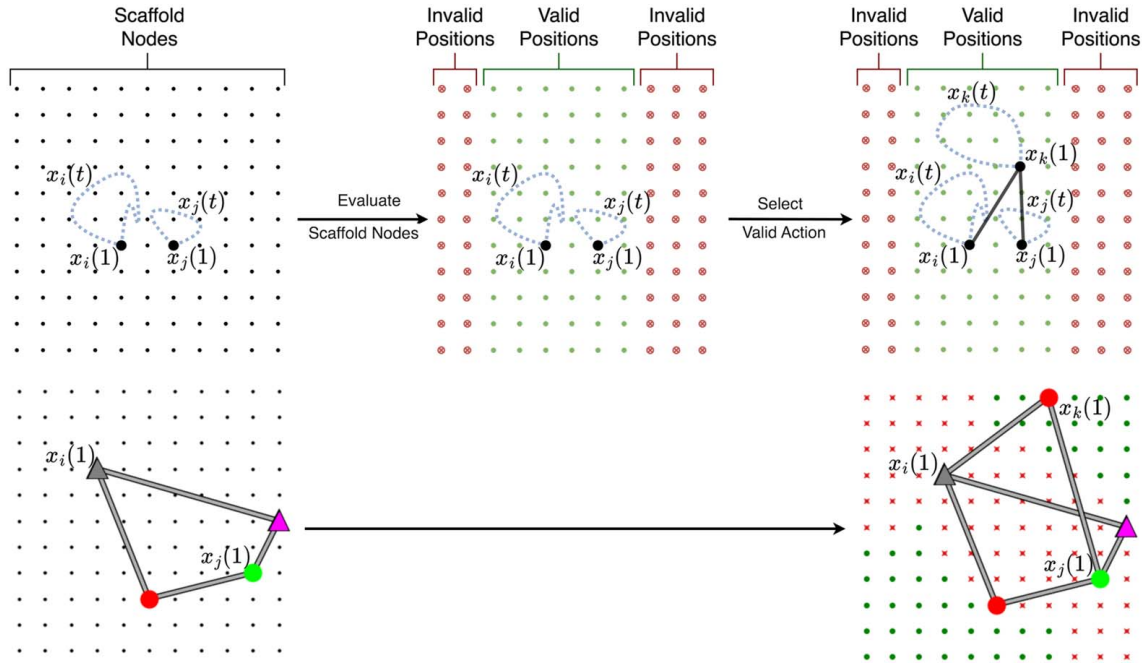


Fig. 4 Graph grammar rule with scaffold nodes and simplified constraint satisfaction criteria

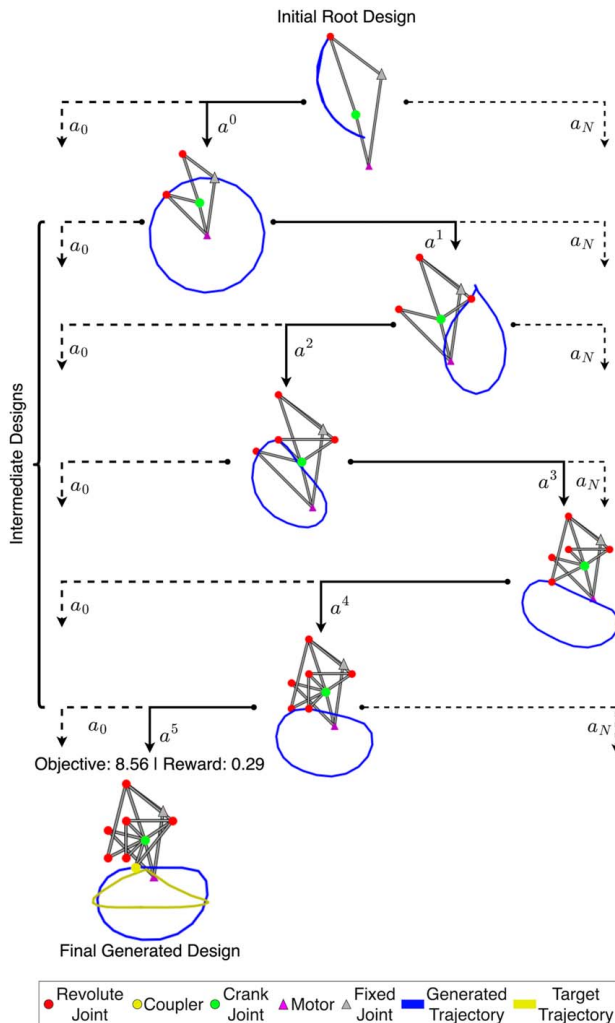


Fig. 5 Linkage design as discrete tree search

that this paper seeks to optimize. To compare the coupler trajectory with the goal trajectory, x_N^* , the average norm squared distance between the two curves is used. This work shifts and scales the coupler trajectory with respect to the goal trajectory. First, all points on both trajectories are shifted by their respective mean, centering the curve at the origin. The points on the coupler curve are scaled by the ratio of the maximum standard deviation, σ , in the x or y direction of the goal curve, σ_N^* , over the maximum standard deviation of the coupler curve, $\max(\sigma_x, \sigma_y)$. This scales all the points on the coupler curve to match the scaling of the goal curve while maintaining the aspect ratio of the path. The normalization can be seen in Eq. (4)

$$x'_N(t) = \frac{x_N(t) - \mu}{\max(\sigma_x, \sigma_y)} \sigma_N^* \forall t \in [1, T] \quad (4)$$

Here $x_N(t)$ is a point on the coupler trajectory at time t , μ is the mean position of all the points in the trajectory, σ_x, σ_y are the standard deviation of the points in the trajectory in the x and y -axis, respectively, σ_N^* is the goal trajectory scaling factor, and finally, $x'_N(t)$ is the normalized position of the point on the trajectory. The normalization is useful since the designer may elect to shift and scale the final mechanism for their specific application. For example, in Ref. [1], the linkages defining the trajectories for the various points on the leg are configured like a gear train with the path generated by the linkages transferred using rigid connectors attached to the coupler node. The work does not consider rotation

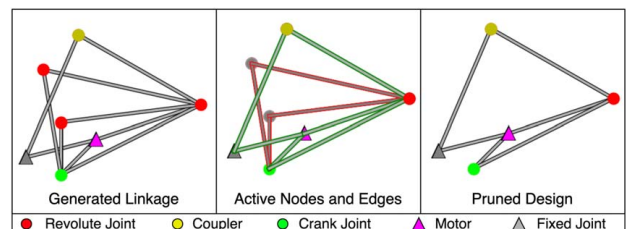


Fig. 6 Pruning generated linkage of unused joints and edges

invariance when comparing the two trajectories, but this could be explored in future research.

The goal trajectory, x_N^* , and generated trajectory, x_N' , are made up of discrete ordered points, however, their mutual ordering is not initially known. All combinations are evaluated to determine the mutual ordering and properly compare the coupler and goal trajectories. There are $2T$ orderings, T orderings by shifting the generated trajectory indices, and T orderings by reversing and shifting the generated trajectory indices; T is the number of discrete sample points. The minimum distance of all ordering combinations is used as the objective value for the generated linkage. The ordering of the points was significant to the algorithm's success in generating meaningful linkages. This process is given in Eq. (5), note that this is the same distance measure used by Pan et al. [11]

$$d_{coupler} = \min_{o_j(t) \in O_j} \frac{2\pi}{T} \sum_{i=1}^T \|x_N'(o_j(t)) - x_N^*(o_i(t))\|^2$$

$$\forall O_j \in [[1 \dots T], [2 \dots 1], \dots, [T \dots 1], [T-1, \dots, T]],$$

$$o_i(t) = [1 \dots T], \quad (5)$$

where d is the objective value of the properly ordered coupler and goal trajectories, o is the index ordering, and T is the number of discrete points that make up the discrete trajectories.

This work uses a sparse reward structure that is only applied to the terminal action. An obvious reward for training this system would be to use the negative distance described in Eq. (5), $-d_{coupler}$. However, using the negative distance of the coupler and goal trajectory is insufficient to find robust solutions because the valuable updates are overwhelmed by the common subpar trajectories generated. This noise inhibits the agent's performance and requires a more thoughtful reward function. A baseline performance between the goal trajectory and the motor input, d_{circle} , normalizes the coupler performance. As the coupler trajectory gets closer to the

goal trajectory, $d_{coupler} \rightarrow 0$, the terminal design receives a reward of one. When the coupler trajectory performs much worse than the baseline, $d_{coupler} \rightarrow \infty$, the reward of the terminal design tends towards $-\infty$, but is clipped to zero. For all actions that lead to invalid designs, such as those that go through singularities, the action is rejected, the previous graph is returned, and the agent receives a reward of -1 . The reward function can be seen in Algorithm 3.

Algorithm 3 Normalize reward

Require: $x'_{coupler}, x'_{goal}, x'_{circle}$	▷ The normalized coupler trajectory, goal trajectory, and circle distance
$d_{circle} \leftarrow \ x'_{circle} - x'_{goal}\ $	▷ Distance between normalized circle and goal
$d_{coupler} \leftarrow \ x'_{coupler} - x'_{goal}\ $	▷ Distance between normalized coupler and goal
$R \leftarrow \max\left(\frac{d_{circle} - d_{coupler}}{d_{circle}}, 0\right)$	▷ Normalize the reward between 0 and 1
return R	

3.2.4 GCP-HOLO Training. GCP-HOLO trains a reinforcement learning agent to explore the tree of valid linkage designs defined by the action space from Sec. 3.2.2. The root node, or initial start design, for the tree search must, at minimum, contain the location for the motor input node, a fixed node, and the node connected to the motor, crank node, a priori. This can be set by the designer or randomly selected. The root node for the search can be fixed or vary at each iteration to better explore the possible set of designs. In this implementation, 100 trees are searched simultaneously with various four-bar root nodes to achieve the reported results. The four-bar root nodes are made up of Grashof crank-rocker and double-rocker linkages, which all satisfy full revolutions. The reinforcement learning agent, through training, learns a stochastic search direction, $\pi(als)$, which is a probability distribution of choosing action a given the current state s . This learned distribution improves the solutions generated over a pure random search. Throughout training, the network sees intermediate states of N-bar linkages, allowing inference to occur at any intermediate linkage design state without retraining. The two reinforcement learning network architectures this paper explicitly compares are PPO [30] and DQN [31], the architecture for these can be seen in Fig. 7. This work uses the policy implementations from OpenAI's Stable-Baselines [32] a well-maintained repository of many common RL algorithms that leverages the parallelization of GPUs.

The reinforcement learning policy, $\pi(als)$, is trained using a history of experience rollouts containing the linkage state, chosen action, reward, and updated linkage state. In the DQN, a long history is kept, and a subset is randomly selected to update the network, approximating the long-term expected reward from the state-action pair known as the Q-value. The SoftMax operation is applied to the Q-values of a given state and all possible actions, which scales the values into a probability distribution of choosing the actions a . In the PPO, there are two networks, a value network that predicts the expected reward and a policy network that predicts a probability distribution over the actions. This algorithm uses a set of the most recent trajectories to update the policy and value networks by comparing the predicted and true values from the rollouts. The actions are selected non-deterministically from the probability distribution generated by the action network. To avoid overfitting in the graph network, several batch normalization layers are included and a tuned learning rate with a linear learning rate scheduler is used. To avoid overfitting in the reinforcement learning networks, a tuned epsilon clipping value for updating the network parameters and a tuned entropy coefficient that encourages variance in the network are

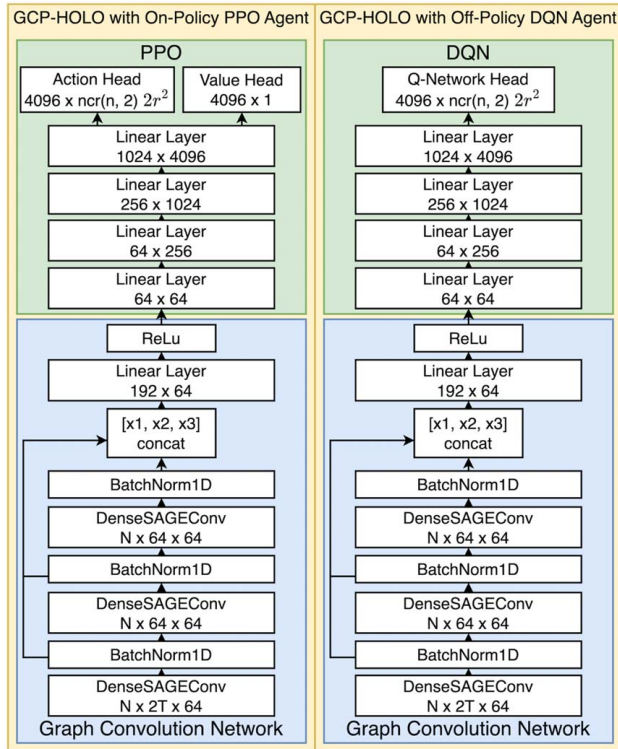


Fig. 7 GCP-HOLO model architecture with on-policy PPO head or off-policy DQN head

used. The complete GCP-HOLO training procedure is shown in Algorithm 4.

Algorithm 4 Training procedure

Require: \mathbf{x}'_{goal} , res , $maxNodes$, $bound$ \triangleright The normalized goal trajectory, resolution, max nodes, and boundary

$bestDesigns \leftarrow \{\}$ \triangleright Initialize best designs as empty dict

$env \leftarrow \text{InitializeEnv}(\mathbf{x}'_{goal}, res, maxNodes, bound)$ \triangleright Initialize Env

$\theta_{Agent} \leftarrow \text{InitializeAgent}()$ \triangleright Initialize Agent

for training steps = 1, 2, ..., **do**

$env.reset()$ \triangleright Sample initial state from distribution

$Buffer \leftarrow \text{CollectRolloutData}(env, \theta_{Agent})$ \triangleright Collect state, action, reward for training

$bestDesigns \leftarrow \text{UpdateBestDesigns}(Buffer)$ \triangleright Update best designs

if training step % update frequency == 0 **then**

for iter = 1, 2, ..., **do**

$\theta_{Agent} \leftarrow \text{UpdateAgent}(\theta_{Agent}, Buffer)$

end for

end if

end for

return θ_{Agent} , $bestDesigns$

GCP-HOLO requires the end-user to input the target trajectory, the resolution of the scaffold nodes, the maximum number of nodes, and any design bounds/constraints. The GCP-HOLO algorithm outputs the learned parameters of the neural network models as well as a set of best designs found during training.

4 Results

This paper evaluates GCP-HOLO through three experiments. In Sec. 4.1, designs generated by GCP-HOLO on eight test trajectories are compared, with respect to objective value and computation time, to designs generated by Pan et al. [11]. Section 4.2 shows an example of a design application with additional body and coupler constraints using GCP-HOLO. Finally, Sec. 4.3 provides an ablation study focused on the efficacy of two reinforcement learning algorithm approaches with respect to the breadth of designs and the solution performance. All experiments were run on 1CPU of Intel Xeon Gold 6252 2.10 GHz and used an Nvidia Tesla V100 GPU. The comparative results from Pan et al. were run in that work on a 10-core Intel Xeon(R) W-2155 CPU using all 10-cores [11]. The hyperparameters used for all experiments are listed in Table 1.

4.1 GCP-HOLO Versus MICP. Eight test curves shared by Pan et al. [11] are used to compare GCP-HOLO performance and their MICP formulation. The objective values of the best designs generated from each of the methods are reported in Fig. 8. A visual of the designs generated by GCP-HOLO and GCP-HOLO with CMA-ES is shown in Fig. 9. For the results shown, GCP-HOLO uses a PPO reinforcement learning agent, trained on 400K steps, which took 35 min. GCP-HOLO sampled 100K designs non-deterministically using the learned policy to generate the solution found; this process took 10 min. The solutions found are at best local minima as GCP-HOLO restricts the initial node

location to the discrete scaffold node. To significantly improve the solutions' performance, an extension that searches other nearby node locations is done using CMA-ES. This extension took less than a minute to converge for each results shown in Figs. 8 and 9.

Figure 8 shows that some solutions generated by GCP-HOLO improve the performance with a 13X speed enhancement over the other approaches. GCP-HOLO found significantly better solutions than all the simulated annealing baselines by Pan et al. on all eight test curves. GCP-HOLO found a better solution from the MICP formulation on two out of the eight test curves with a run time improvement from the reported 10+ h to 45 min. With the extension of CMA-ES global dimensional synthesis, four out of the eight solutions outperform the MICP and mixed integer non-linear program (MINLP) designs, which Pan et al. reported taking 5–10 h for MINLP to generate or 10+ h for MICP to generate [11]. The drastic improvement through the global optimization with CMA-ES shows how sensitive the linkage design problem is. The 13X speed improvement seen by GCP-HOLO is in part due to the efficiency of only generating constraint-satisfying linkages and its ability to leverage the GPU. The biggest downside to the MICP method is that before the method converges, the current best design may not be a valid linkage. Pan et al. [11] show that most of the time of the algorithm is spent trying to find a valid linkage design. In this work, however, by adding the 0DOF linkages to grow the linkage graph with the spatial action constraint, it is easy to generate constraint-satisfying linkages. With the improved efficiency and performance in some instances, GCP-HOLO addresses some of the limitations of Pan et al. [11] like the linkage generation of high-order linkage graphs and computational complexity.

4.2 Path Synthesis Application. In many situations—e.g., designing linkage walking systems, pick and place mechanisms, or construction cranes—it may be necessary to constrain the coupler node in a specific region. In the example here, with results shown in Fig. 10, an application of designing a walking linkage is shown by generating linkages that satisfy the Trotbot trajectory [33]. A constraint on the coupler node location in the lower plane and the other node locations in the upper plane was added. The output of GCP-HOLO shows the best designs found for various linkage graph complexities. The solutions generated by GCP-HOLO found designs that minimize the objective and satisfy the constraints with linkage graphs that contain 1–5 cycles. The best design found had three active cycles or an eight-bar linkage.

The top of Fig. 10 shows the solutions generated by an unconstrained GCP-HOLO, which unsurprisingly does not satisfy the constraint condition. The designs generated from the unconstrained GCP-HOLO better minimize the distance to the goal trajectory. In one case, the constrained GCP-HOLO found a better design than the unconstrained GCP-HOLO. One explanation is that the GCP-HOLO method searches the set of designs using only valid actions. Adding the extra constraint of the node locations decreases the number of valid actions, thus decreasing the total number of designs in the search tree. With a smaller search tree, a more significant proportion of designs can be explored, and better solutions found.

Table 1 Hyperparameters for GCP-HOLO used in all experiments

Values	Max nodes	Scaffold node resolution	Trajectory sample points	Learning rate	Batch size	Number envs	Number training steps	Number eval episodes	Sigma CMA-ES	Tol CMA-ES
	11	11	20	1×10^{-4}	1×10^4	100	4×10^5	1×10^5	5×10^{-5}	1×10^{-4}

Input	GCP-HOLO	GCP-HOLO + CMA-ES	MINLP	MICP	SA-Baseline $i_{\max} = 5 \times 10^4$	SA-Baseline $i_{\max} = 5 \times 10^5$
	Time: 45min	Time: 46min	Time: 5-10 hrs	Time: 10+hrs	Time: 1hr	Time: 10hrs
	1.09	0.44	0.7	0.77	6.2	7.75
	3.29	0.87	1.36	1.36	4.49	7.85
	3.45	2.73	0.71	2.58	4.96	5.35
	0.79	0.32	1.25	1.37	16.15	4.57
	1.78	1.58	1.39	2.52	2.66	6.8
	1.25	0.38	0.36	0.32	15.88	15.8
	0.61	0.15	0.52	0.51	10.03	11.09
	2.25	1.07	0.77	1.28	7.9	9.67

Fig. 8 Comparison of best design objective value (lower is better) from GCP-HOLO, GCP-HOLO with CMA-ES on eight test trajectories designed by Pan et al. [11]. Results from MICP, MINLP, and simulated annealing were taken from Pan et al. [11].

4.3 On-Versus Off-Policy Comparison. Any policy can generate a variety of linkages using the GCP-HOLO action space, but the breadth of solutions that span various linkage topologies may be limited. Designs that span different linkage graph topologies enable better starting points for further exploration and assessment. In this ablation study, a trained GCP-HOLO model with an on-policy PPO policy, a trained GCP-HOLO model with an off-policy DQN policy, and a random search each generate 20 trials of 100K linkage designs. All policies are trained with the Jansen linkage trajectory set as the goal [24]. The designs generated are pruned and separated into groups based on the number of active cycles that comprise the linkage graph. The distributions of the designs are evaluated on the number of high-order linkage graphs that were

generated and given in Table 2, the minimum distance measured (the best design) presented in Table 3, the mean distance measured given in Table 4, and the number of invalid linkages generated and shown in Fig. 11. The average over the 20 trials and the standard error are presented in each table.

Table 2 shows that GCP-HOLO with PPO generates the most high-order linkage graphs. In some high-order linkage graphs, GCP-HOLO with DQN generates more designs; this is not unexpected as the policies are stochastic. By generating more high-order linkage graphs, it is more likely that GCP-HOLO will find better designs for minimizing the distance to the goal. PPO found significant improvement in the average best design generated over the other approaches in five of the six high-order linkage graph types









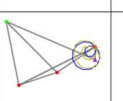
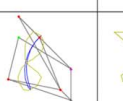
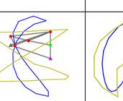
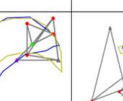
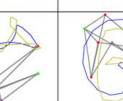
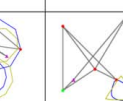
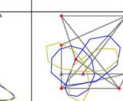

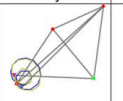
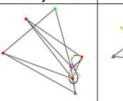
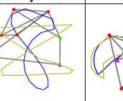
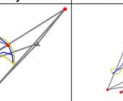
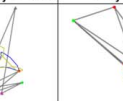
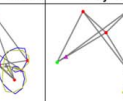
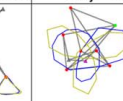

Input								
GCP-HOLO	 Obj = 1.09	 Obj = 3.29	 Obj = 3.45	 Obj = 0.79	 Obj = 1.78	 Obj = 1.25	 Obj = 0.61	 Obj = 2.25
GCP-HOLO + CMA-ES	 Obj = 0.44	 Obj = 0.87	 Obj = 2.73	 Obj = 0.32	 Obj = 1.58	 Obj = 0.38	 Obj = 0.15	 Obj = 1.07

Fig. 9 Best designs generated from GCP-HOLO, GCP-HOLO with CMA-ES on eight test trajectories designed by Pan et al. [11]

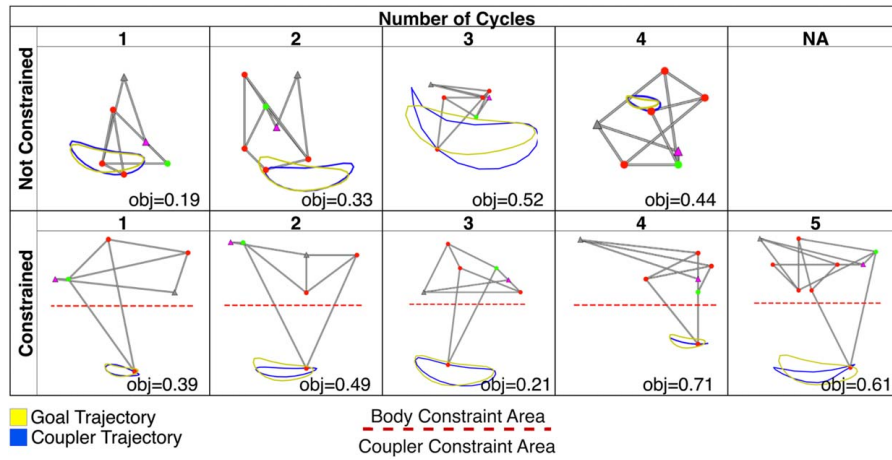


Fig. 10 Designs generated by unconstrained GCP-HOLO and constrained GCP-HOLO with TrotBot goal trajectory

Table 2 Comparison of GCP-HOLO with on-policy PPO policy, GCP-HOLO with off-policy DQN policy, and random policy on the number of designs generated in various linkage graph topologies

Cycles	1	2	3	4	5	6	7	8
PPO	27,343 \pm 131	28,541 \pm 127	28,466 \pm 107^a	11,133 \pm 89^a	2445 \pm 45^a	375 \pm 20	20 \pm 4	1 \pm 0.7
DQN	23,509 \pm 225	32,884 \pm 197^a	21,005 \pm 149	8511 \pm 87	1920 \pm 39	560 \pm 28^a	31 \pm 5^a	1 \pm 0.6
Random	29,886 \pm 129^a	32,171 \pm 186	15,251 \pm 83	4002 \pm 52	675 \pm 23	95 \pm 8	7 \pm 2	0 \pm 0.5

^aStatistically significant.

Table 3 Comparison of GCP-HOLO with on-policy PPO policy, GCP-HOLO with off-policy DQN policy, and random policy on the minimum error of designs generated in various linkage graph topologies

Cycles	1	2	3	4	5	6	7	8
PPO	0.44 \pm 0.007	0.43 \pm 0.011	0.28 \pm 0.014	0.38 \pm 0.014	0.47 \pm 0.015^a	0.62 \pm 0.029^a	1.63 \pm 0.180	5.15 \pm 0.63^a
DQN	0.35 \pm 0.005^a	0.34 \pm 0.007^a	0.27 \pm 0.009	0.39 \pm 0.030	0.54 \pm 0.031	0.68 \pm 0.057	1.60 \pm 0.167	7.33 \pm 2.52
Random	0.38 \pm 0.011	0.38 \pm 0.016	0.32 \pm 0.014	0.44 \pm 0.021	0.61 \pm 0.027	0.99 \pm 0.077	2.822 \pm 0.29	7.62 \pm 3.15

^aStatistically significant.

in Table 3. These designs represent the ones with the best performance with respect to the objective function. This shows that the PPO agent is more likely to find designs that better satisfy the primary objective. Table 4 shows the average performance of all designs generated by the various approaches. Table 4 excludes values reported for instances where less than ten designs are generated because the statistical methods are not indicative of distribution. Table 4 shows that, on average, the designs generated by GCP-HOLO with PPO agent were better than other approaches. These distributions are important to consider since there may be objectives that are not explicit in the optimization, such as

manufacturability and aesthetics. Generating sets of designs improves the end-user selections. Finally, Fig. 11 shows that the PPO agent generated the fewest number of invalid designs. While the scaffold node simplified constraint satisfaction criteria shown in Algorithm 1 reduces the number of invalid actions, it does not prevent them altogether. Both DQN and PPO reduce the number of invalid designs generated, but PPO far reduces that number in comparison to the other approaches.

This comparison between the GCP-HOLO model with PPO policy, the GCP-HOLO model with DQN policy, and random search shows that the learned search heuristic from both

Table 4 Comparison of GCP-HOLO with on-policy PPO policy, GCP-HOLO with off-policy DQN policy, and random policy on the average error of designs generated in various linkage graph topologies

Cycles	1	2	3	4	5	6	7	8
PPO	7.1 \pm 0.01	7.1 \pm 0.01^a	6.8 \pm 0.01^a	7.4 \pm 0.01^a	7.5 \pm 0.03^a	7.8 \pm 0.07^a	8.4 \pm 0.41	—
DQN	7.1 \pm 0.01	8.7 \pm 0.01	7.5 \pm 0.01	8.5 \pm 0.02	8.8 \pm 0.03	8.3 \pm 0.04	7.1 \pm 0.25	—
Random	7.1 \pm 0.01	8.0 \pm 0.01	7.8 \pm 0.01	7.9 \pm 0.02	7.9 \pm 0.05	8.2 \pm 0.15	—	—

^aStatistically significant.

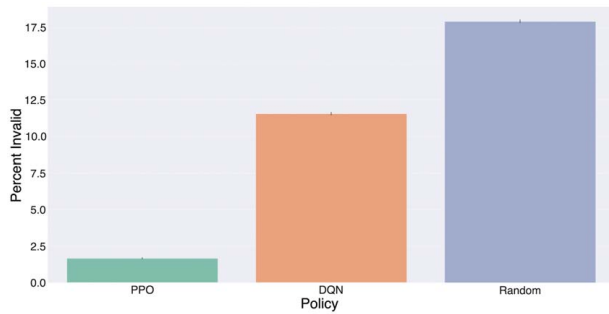


Fig. 11 Average number of invalid designs generated for each policy

PPO and DQN generates better designs than random search. It also presents several advantages of the PPO policy over the DQN policy, such as the breadth of designs and performance of solutions. For these reasons, using GCP-HOLO with the PPO policy is recommended.

5 Discussion

GCP-HOLO uses a learned stochastic search approach for generating constraint satisfying high-order linkage graphs for the inverse design of linkages. Section 4.1 shows that the solutions generated by GCP-HOLO achieved a lower target trajectory distance value on 25% of the test trajectories compared to the MICP formulation while reducing the computational complexity by 13X. The solutions outperform the state-of-the-art method in 50% of the test cases when using the generated designs as initial positions for further optimization using CMA-ES. Section 4.2 shows an application of GCP-HOLO with additional constraints, generating designs with varying linkage graph complexity. The breadth of design solutions can be a starting point to explore further by the end-user, such as evaluating aesthetics or manufacturability. Section 4.3 characterizes two reinforcement learning policies that can be used in GCP-HOLO for this problem. Section 4.3 demonstrates that PPO generates significantly more high-order linkage designs, generates the designs with the best performance, improves design distribution performance, and reduces the number of invalid designs susceptible to this method.

While GCP-HOLO presents improvements to the complex path synthesis problem for generating linkages that satisfy the desired kinematic behavior, several limitations still exist. First, since GCP-HOLO is a stochastic search algorithm, there are no guarantees this paper can make about optimality in a finite time horizon, but similar methods have been shown to generate optimally directed solutions in multi-modal discontinuous problems [20,22]. Next, the dimensionality of the linkages generated should be able to satisfy the path synthesis problem better. Still, the current discrete action space strongly limits the performance of the method due to the sensitivity of the dimensional synthesis problem. One approach could be to extend GCP-HOLO through a bi-level process where GCP-HOLO proposes constraint satisfying initial states and secondary optimization methods further improve the solution. Another interesting extension would be to incorporate relaxations to the discretization of the design spaces shown in Raina et al. [34], to generate new node locations that are not confined to the scaffold node grid. The topological rule used in this work cannot generate all topological variants; understanding which topologies are suited for the desired path and finding new topological rules to span all linkage topologies is an interesting direction. Another key limitation is the lack of manufacturability considerations when generating linkages with GCP-HOLO. To further support non-experts in designing linkages, manufacturability, link conflicts, stresses, and end effector dynamics must be considered.

GCP-HOLO currently only supports 1DOF planar linkage design, but spatial linkage design could be considered using a similar generation technique. The hardest challenge is that the action space becomes proportional to r^3 , greatly increasing the branching factor. However, adding 0DOF spatial linkages to grow the linkage graph with scaffold nodes iteratively would enable the design of linkages outside the plane.

6 Conclusion

GCP-HOLO is an RL-based tree search method that leverages a graph expansion rule and recursive kinematics to efficiently search for high-order linkage graphs that satisfy the path synthesis problem. Compared to other state-of-the-art methods, GCP-HOLO performs 13X faster. GCP-HOLO learns a stochastic search policy to find solutions by exploring the tree of valid linkages. The solutions generated by GCP-HOLO can be extended with geometric optimization of the node positions, which results in designs that perform the best on 50% of the test cases. This paper shows an application of designing a linkage for a walking system using GCP-HOLO that satisfies additional constraints related to the body and coupler positions. GCP-HOLO is a general method for path synthesis which has applications in many areas from automotive, space, and construction, where linkages play a role in their systems. This paper showed that PPO, an on-policy agent, was advantageous to DQN, an off-policy agent, for this problem. This paper presents a method that reduces the end user's domain expertise needed by generating sets of solutions of varying linkage graph complexity, extending the capabilities of design systems and mechanical designers in creating future design solutions.

Acknowledgment

The authors thank Ayush Raina, Sakthi Prakash, James Cunningham, and Josh Gyory for their discussion on this work. The authors want to thank Pan et al. for sharing the test curves they created.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The datasets generated and supporting the findings of this article are obtainable from the corresponding author upon reasonable request.

Nomenclature

CMA-ES	= covariance matrix adaptation evolutionary strategy
DOF	= degrees-of-freedom
DQN	= deep Q-network
FK	= forward kinematics
GCP-HOLO	= graph convolution policy for high-order linkage graph optimization
GNN/GCN	= graph neural network/graph convolution network
MICP	= mixed integer conic program
PPO	= proximal policy optimization
RL	= reinforcement learning

References

- [1] Plecnik, M. M., and McCarthy, J. M., 2016, "Computational Design of Stephenson II Six-Bar Function Generators for 11 Accuracy Points," *ASME J. Mech. Rob.*, 8(1), p. 011017.
- [2] Ramezani, A., Shi, X., Chung, S. J., and Hutchinson, S., 2016, "Bat Bot (B2), A Biologically Inspired Flying Machine," *Proceedings—IEEE International*

- Conference on Robotics and Automation, Stockholm, Sweden, May 16–21, pp. 3219–3226.
- [3] Festo USA, 2018, “BionicFlyingFox,” https://www.festo.com/us/en/e/about-festo/research-and-development/bionic-learning-network/highlights-2018/bionic-flyingfox-id_32755/?siteUId=fox_us&siteName=Festo+USA
 - [4] Plecnik, M. M., Haldane, D. W., Yim, J. K., and Fearing Professor, R. S., 2017, “Design Exploration and Kinematic Tuning of a Power Modulating Jumping Monopod,” *ASME J. Mech. Rob.*, **9**(1), p. 011009.
 - [5] Mruthyunjaya, T. S., 2003, “Kinematic Structure of Mechanisms Revisited,” *Mech. Mach. Theory*, **38**(4), pp. 279–320.
 - [6] Tuttle, E. R., 1996, “Generation of Planar Kinematic Chains,” *Mech. Mach. Theory*, **31**(6), pp. 729–748.
 - [7] Freudenstein, F., 1955, “An Analytical Approach to the Design of Four-Link Mechanisms,” *Trans. ASME*, **76**(3), pp. 483–492.
 - [8] Plecnik, M. M., and Fearing, R. S., 2020, “Designing Dynamic Machines With Large-Scale Root Finding,” *IEEE Trans. Rob.*, **36**(4), pp. 1135–1152.
 - [9] Lipson, H., 2008, “Evolutionary Synthesis of Kinematic Mechanisms,” *AI EDAM*, **22**(3), pp. 195–205.
 - [10] Vermeer, K., Kuppens, R., and Herder, J., 2018, “Kinematic Synthesis Using Reinforcement Learning,” *ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Quebec City, Canada, Aug. 26–29.
 - [11] Pan, Z., Liu, M., Gao, X., and Manocha, D., 2022, “Joint Search of Optimal Topology and Trajectory for Planar Linkages,” *Int. J. Rob. Res.*
 - [12] Assur, L. V., 1914, “Investigation of Plane Hinged Mechanisms With Lower Pairs From the Point of View of Their Structure and Classification (in Russian): Part I, II,” *Bull. Petrograd Polytech. Inst.*, **21**, pp. 187–283.
 - [13] Wampler, C. W., Morgan, A. P., and Sommese, A. J., 1992, “Complete Solution of the Nine-Point Path Synthesis Problem for Four-Bar Linkages,” *ASME J. Mech. Des.*, **114**(1), pp. 153–159.
 - [14] Plecnik, M. M., and McCarthy, J. M., 2015, “Design of Stephenson Linkages That Guide a Point Along a Specified Trajectory,” *Mech. Mach. Theory*, **96**, pp. 38–51.
 - [15] Thomaszewski, B., Coros, S., Gauge, D., Megaro, V., Grinspun, E., and Gross, M., 2014, “Computational Design of Linkage-Based Characters,” *ACM Trans. Graph.*, **33**(4), pp. 1–9.
 - [16] Bächer, M., Coros, S., and Thomaszewski, B., 2015, “Linkedit: Interactive Linkage Editing Using Symbolic Kinematics,” *ACM Trans. Graph.*, **34**(4), pp. 1–8.
 - [17] Zhao, P., Ge, X., Zi, B., and Ge, Q. J., 2016, “Planar Linkage Synthesis for Mixed Exact and Approximated Motion Realization Via Kinematic Mapping,” *ASME J. Mech. Rob.*, **8**(5), p. 051004.
 - [18] Nanni, L., Ghidoni, S., and Brahnam, S., 2017, “Handcrafted vs. Non-Handcrafted Features for Computer Vision Classification,” *Pattern Recogn.*, **71**, pp. 158–172.
 - [19] Raina, A., McComb, C., and Cagan, J., 2019, “Learning to Design From Humans: Imitating Human Designers Through Deep Learning,” *ASME J. Mech. Des.*, **141**(11), p. 111102.
 - [20] You, J., Liu, B., Ying, R., Pande, V., and Leskovec, J., 2018, “Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation,” *Conference on Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, Dec. 3–8.
 - [21] Whitman, J., Bhirangi, R., Travers, M., and Choset, H., 2020, “Modular Robot Design Synthesis With Deep Reinforcement Learning,” *Proceedings of the AAAI Conference on Artificial Intelligence*, New York, Feb. 7–12.
 - [22] Zhao, A., Xu, J., Konaković-Luković, M., Hughes, J., Spielberg, A., Rus, D., and Matusik, W., 2020, “RoboGrammar: Graph Grammar for Terrain-Optimized Robot Design,” *ACM Trans. Graph.*, **39**(6), pp. 1–16.
 - [23] Raina, A., Puentes, L., Cagan, J., and McComb, C., 2021, “Goal-Directed Design Agents: Integrating Visual Imitation With One-Step Lookahead Optimization for Generative Design,” *ASME J. Mech. Des.*, **143**(12), p. 124501.
 - [24] Jansen, T., 1990, “Strandbeest,” <https://www.strandbeest.com/>
 - [25] Kavitha, T., Mehlhorn, K., Michail, D., and Paluch, K. E., 2007, “An $O(m^2n)$ Algorithm for Minimum Cycle Basis of Graphs,” *Algorithmica*, **52**(3), pp. 333–349.
 - [26] Kecskeméthy, A., Krupp, T., and Hiller, M., 1997, “Symbolic Processing of Multiloop Mechanism Dynamics Using Closed-Form Kinematics Solutions,” *Multibody Syst. Dyn.*, **1**(1), pp. 23–45.
 - [27] Tsai, W., and McCarthy, J. M., 2000, “Mechanism Design: Enumeration of Kinematic Structures According to Function,” *ASME J. Mech. Des.*, **122**(4), pp. 583–583.
 - [28] Kipf, T. N., and Welling, M., 2017, “Semi-Supervised Classification With Graph Convolutional Networks,” *5th International Conference on Learning Representations, ICLR 2017—Conference Track Proceedings*, Toulon, France, Apr. 24–26.
 - [29] Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y., 2019, “Invariant and Equivariant Graph Networks,” *The International Conference of Learning Representation (ICLR)*, <https://openreview.net/forum?id=Syx72jC9tm>, Accessed July 3, 2022.
 - [30] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Openai, O. K., 2017, “Proximal Policy Optimization Algorithms,” *arXiv preprint*.
 - [31] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., et al., 2015, “Human-Level Control Through Deep Reinforcement Learning,” *Nature*, **518**(7540), pp. 529–533.
 - [32] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N., 2021, “Stable-Baselines3: Reliable Reinforcement Learning Implementations,” *J. Mach. Learn. Res.*, **22**(268), pp. 1–8.
 - [33] Vagle, W., 2015, “Team Trotbot,” <https://www.teamtrotbot.com/>, <https://www.teamtrotbot.com/mechanism.html>, Accessed April 21, 2022.
 - [34] Raina, A., Cagan, J., and McComb, C., 2022, “Design Strategy Network: A Deep Hierarchical Framework to Represent Generative Design Strategies in Complex Action Spaces,” *ASME J. Mech. Des.*, **144**(2), p. 021404.