Kinematic synthesis using reinforcement learning

Vermeer, Kaz; Kuppens, Reinier; Herder, Just

DOI
10.1115/DETC2018-85529

Publication date
2018

Document Version
Final published version

Published in
ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference

**Proceedings of the ASME 2009 International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
IDETC/CIE 2018
August 26-29, 2018, Quebec City, Canada**

n/a

KINEMATIC SYNTHESIS USING REINFORCEMENT LEARNING

K.M. (Kaz) Vermeer
Faculty 3mE
Dep. Precision and
Microsystems Engineering
Delft University of Technology
Delft, 2628 CD
The Netherlands

P.R. (Reinier) Kuppens *
Faculty 3mE
Dep. Precision and
Microsystems Engineering
Delft University of Technology
Delft, 2628 CD
The Netherlands
P.R.Kuppens@tudelft.nl

J.L. (Just) Herder
Faculty 3mE
Dep. Precision and
Microsystems Engineering
Delft University of Technology
Delft, 2628 CD
The Netherlands
J.L.Herder@tudelft.nl

ABSTRACT

The presented research demonstrates the synthesis of two-dimensional kinematic mechanisms using feature-based reinforcement learning. As a running example the classic challenge of designing a straight-line mechanism is adopted: a mechanism capable of tracing a straight line as part of its trajectory. This paper presents a basic framework, consisting of elements such as mechanism representations, kinematic simulations and learning algorithms, as well as some of the resulting mechanisms and a comparison to prior art. A sensitivity analysis analyzes the neural network's considerations with respect to the selected features. Series of successful mechanisms have been synthesized for path generation of a straight line and a figure-eight.

INTRODUCTION

Mechanism synthesis can be an arduous task, often as much art as science. Therefore, support from computational power in exploring the solution space can be of great value. Research into such computer-aided mechanism synthesis mainly focuses on topology optimization for compliant mechanisms [1] or evolutionary algorithms for rigid body mechanism [2–4].

None of these efforts however apply reinforcement learning (RL), an experience-based learning concept currently applied successfully in other fields [5–7]. RL is specifically well-suited for game-like situations [8] in which decisions have to be made and (delayed) rewards may be obtained. Application of RL onto mechanism design therefore requires the development of an inventive new framework in which design challenges are posed in a game-like fashion.

The present paper reports a method for formulating kinematic synthesis of rigid body mechanisms as a RL problem using a decision-tree-based mechanism representation and nonlinear value function approximation in an effort to make mechanism design a game-like process. To demonstrate the method's effectiveness several straight-line mechanism are synthesized. Using the same algorithm and parameter settings, a mechanism for generating figure-eight patterns is synthesized. Additionally a sensitivity analysis is performed to investigate what features are valued most by the learning algorithm. Finally a benchmark study is performed.

First, the method used in this research is presented after which the experiment is described. Hereafter the results from the experiment are discussed. The paper ends with a discussion and conclusion.

*Address all correspondence to this author.

METHOD

Mechanism representation

In order to synthesize, interpret, evaluate and manipulate mechanism designs, a numeric system, i.e. a computer, has to be able to communicate about such objects. This requires a befitting language, or in formal terms a numeric representation of designs. Several options depend on a discretized solution space (e.g. building blocks [9] truss-based [10] and density methods [4]). However, discretization introduces unwanted dependence on designer inputs like mesh-size or building block selection. Alternatively, graph-based representations can be used to describe mechanisms. A particularly convenient method is the decision-tree, since reinforcement learning is especially adept at sequential decision making. And, as shown by [2], decision-operators can be chosen such that the solution space is limited to the feasible domain.

Lipson [2] proposed a decision-tree representation to describe a series of decisions on a seed mechanism resulting in a unique kinematic mechanism. In each decision one of two operations is performed: T or D. Given a kinematic system of interconnected links and hinges, both operators act on a specified target link when executed. The operators introduce a new hinge and connect it to the the target link's endpoints. In case of a T operator, the target link is subsequently replaced by a connecting link between the new hinge and its nearest non-connecting neighbor. A D operator simply leaves the target link untouched. Using Fig. 1 one may compare the different results of these operators.

Both T and D have the particularly useful property of conserving a mechanism's number of degrees of freedom (DOF). For our path generation objectives, only 1-DOF systems are desired. So, given a 1-DOF seed mechanism, these operator are intrinsically restricting the design space to include only feasible mechanisms. This is in contrast to most other representations, like non-tree graphs, which do allow over-constraint, under-constraint and kinematically determined mechanisms to be synthesized. The sequence of operators can be represented as a decision tree as shown in [2].

In [2], the operators are accompanied by a local coordinate, describing the placement of the new node relative to the target link's position, see Fig. 2a. To refrain from the complexity of continuous action spaces these local coordinates are not used in the method presented in this paper. Instead new nodes are placed automatically such that they form an isosceles triangle with the target link. The resulting triangle's height is set equal to the target link's length and the new node is placed on the outside of the existing mechanism, see Fig. 2b.

Kinematic modeling

Kinematic modeling is required to evaluate a mechanism's path generation accuracy. During the learning process this evaluation procedure is performed hundreds of thousands of times.

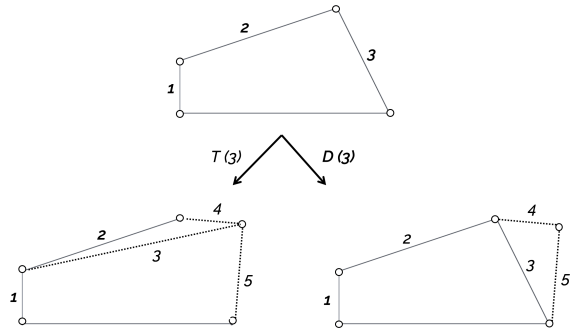


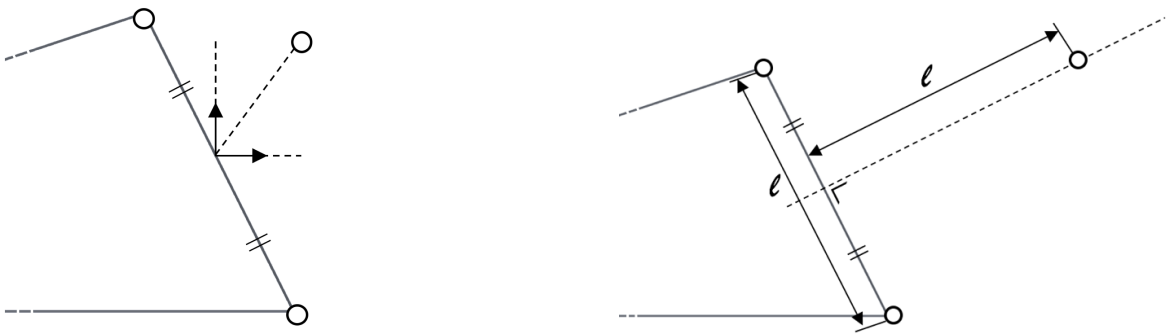
FIGURE 1: Example of different outcomes for D and T operations on link number 3. The full lines represent links, the circles represent hinges and the dotted lines represent newly created links.

Therefore, a robust, general and computationally cheap analysis method is desired which can handle any conceivable mechanism within the representation's domain. To do so numeric simulation is used.

Fig. 3 shows the initial configuration of an example mechanism. The bottom two nodes are only allowed rotational motion. The curved arrow indicates the input link. In a simulation the input link is rotated a full circle in 300 equally spaced steps by prescribing its position. After each perturbation the positions of the free nodes are determined by solving the position problem. Finally a sampled trajectory is extracted. Applying this method to the mechanism presented in Fig. 3 leads to the curved trajectory shown by the dashed line.

Solving the position problem The position problem has to be solved for all 300 prescribed positions of the input link. However depending on the geometry of a mechanism, the input link may not be able to turn a full circle through all positions. For such kinematically inadmissible positions there is no valid solution to the position problem. Therefore a method is required that can cope with such situations. [11] presents a method in which mechanisms are modeled using rod-type finite elements with revolute joints on both ends, allowing for axial deformation with unit stiffness. Solutions to the position problem can therefore always be found, even for kinematically inadmissible positions, by deforming some of the links. In each time step the position problem is solved iteratively by determining and minimizing the error function: the system's elastic potential V. Newton's second-order method [12] is adopted to solve the optimization problem, minimizing V with respect to the nodal coordinates x. The error function for a linkage with a total of b links of length L_e equals:

$$V(x) = 1/2 \sum_{e=1}^{b} (l_e(x) - L_e)^2 = 1/2 \sum_{e=1}^{b} \left(\sqrt{x^T [\bar{g}]_e x} - L_e \right)^2$$



(a) (b)

FIGURE 2: Positioning of new nodes based on Lipson's [2] method (a) and the method proposed in this paper (b). Along with the operator type and target link number, Lipson' operators include a local coordinate describing the position of the new nodes relative to the target link. In the proposed method only the operator type and target link are specified. The new node is positioned such that it creates an isosceles triangle with the target link.

in which $l_e(x)$ represents the length of element e as a function of the nodal coordinates in x and $\bar{g}$ is an assembled reduced form stiffness matrix that allows for fast computation. During iteration, l_e might not be equal to L_e, introducing elastic potential and thereby error. Iterative updates of x continue until the error V drops below a predefined tolerance of $1 \cdot 10^{-3}$ or the iteration count exceeds 100.

Kinematically inadmissible positions and singularities During numeric analysis, kinematically inadmissible domain can be detected by high levels of elastic potential. By removing trajectory points with high elastic potential only the feasible domain remains. In the limit cases, i.e. mechanisms with dimensions just slightly preventing it from entering infeasible domain, singularities can occur. In such cases somewhere along the trajectory two or more links become aligned resulting in locking behavior, effectively losing one degree of freedom. Such an unstable mechanism position results in large gradients of the potential energy. Accordingly, the gradient and Hessian matrices in Newton's second-order method become ill-conditioned. To deal with the resulting large gradients an adaptive learning rate is employed, preventing overshoots whilst searching for an equilibrium position. This adaptive learning rate α is cut in half whenever the error is rising instead of falling. No updates of x are performed in this case until the error ceases to rise.

Learning approach

RL is described by Kaelbling et al. [8] as behavioral learning by "trial-and-error interactions with a dynamic environment" RL does not require a priori data, but instead learns from reward signals received after each experienced data point. Hence a machine can learn by taking actions, assessing the resulting rewards and adapting its decision policy accordingly. Through iteration, the machine can learn to take the actions that maximize the total reward.

In RL the learner is named the *actor* [13]. The actor gains experience through taking *actions* that lead to interactions with its *environment*. After every interaction, the actor is faced with a new situation, or *state*, from which it may perform the next action. As part of the interactions, the actor may receive *rewards*. The actor's goal is to choose actions such that it maximizes the total reward. In a more formal manner, one can say that for every time-step t the actor is in state S_t. After taking action A_t, it ends up in state S_{t+1} with probability $P_{S_tA_t}(S_{t+1})$ and receives a reward R_{t+1}. A schematic of this process is shown in Fig. 4. Over time, the actor's total return G_t equals the sum of all future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \qquad (1)$$

A discount factor γ is introduced as a means to promote fast results. This procedure is governed by the five-element tuple (S, A, P, γ, R). Decision processes described by such a tuple are sometimes referred to as Markov Decision Processes (MDP) [14], referring to their adherence to the Markovian property [15] and inclusion of both deterministic and stochastic elements. Using the tuple's elements, one can define the value function $V_\pi(s)$ as the expected return G of being in state s and following policy π until termination [13]:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right] \end{aligned} \qquad (2)$$

Since one is usually concerned with choosing the right action, it is common practice [13, 14] to include both the state and the action as variables of the value-function, leading to the state-action value-function Q:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \end{aligned} \tag{3}$$

For a known MDP, the fixed point theorem [16] shows that V_π can always be found in a recursive fashion. In reality however, the MDP is never fully known and V or Q can only be estimated. Several algorithms exist that can be used to improve an estimation of V or Q after each completed series of states and actions also known as an episode. To improve efficiency, one can bootstrap by updating after each time-step instead of after each episode. The former method is referred to as Monte Carlo learning (MC), whereas the latter is known as Temporal Difference learning (TD). Both methods are described in depth by Sutton and Barto [13].

The approximated Q-function becomes more and more accurate by learning. This is exploited to make sound decisions and therefore maximize reward R_t. Always choosing the action that maximizes expected reward in $t+1$ is known as a greedy policy. Such a policy however is short-sighted and likely to get stuck in local optima. To prevent such behavior a policy called ε-greedy [13, 17, 18] is used. This policy introduces randomly selected actions with a probability of ε to encourage exploration of the action space. In the current application ε is initialized as high as 0.5 and diminishes throughout the learning process.

For the current application TD learning is chosen, since this method is found to generally converge faster then its MC counterpart [13]. The simplest and best-known TD algorithm is SARSA, its pseudocode shown in algorithm 1. In each time-step t an error term between the true and expected reward is determined. Afterwards the approximation of $Q(S_t,A_t)$ is adjusted in the direction of the true value by a small step of size ν.

Value function approximation In the SARSA algorithm $Q(S,A)$ is calculated and adjusted. In a practical sense, this means Q should be an entity that can take a state and action as input and produce an expectation as output. Several embodiments exists for such an entity: closed-form mathematical functions, lookup-tables and (non)linear function approximation.

Closed-form mathematical functions, used to calculate Q as $f(S,A)$, are not a fitting choice as they generally are not compatible with state-descriptions and discrete actions as inputs. Secondly, iterative updates of closed-form functions can only materialize by means of parameter adjustments. Selecting the right parameters to include in the function however requires a priori knowledge of the desired function approximator, which is unavailable.

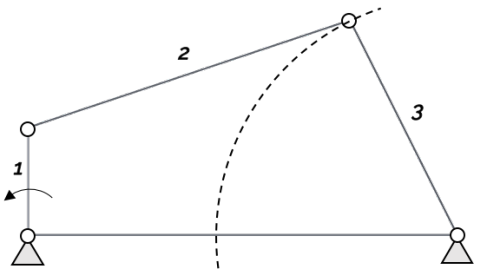


FIGURE 3: The starting point: a basic four-bar mechanism with two grounded nodes. The arrow indicates the prescribed motion on input link 1. Link 2, often named coupler link, transfers the input motion to output link 3. The dashed line depicts the output link's path, moving back and forth along the same trajectory.

Initialization;
  $Q \leftarrow$ arbitrary;
foreach *Episode* **do**
  Initialize S;
  Choose A from S using ε-greedy policy and Q;
  foreach *Step in the episode* **do**
    Take action A, receive R and S';
    Choose action A' from S' using ε-greedy policy and Q;
    $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \nu\left[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)\right]$;
    $S \leftarrow S'$; $A \leftarrow A'$;
  end
end

Algorithm 1: SARSA using TD learning from [13]

Secondly, one could represent $Q(S,A)$ in a tabular fashion, keeping track of a separate Q value for each unique state-action pair [13]. For specific applications this method can be effective [19]. However, besides the obvious memory limitation to small state and action spaces, the tabular approach also fails in what is known as the generalization of states [20]: the ability to use experience gained in state-action pair (S,A) for decision making in closely related state-action pairs. In a table each state-action pair is treated as a perfectly separated individual, such that experience from visiting state-action pair (S,A) is only valuable for future visits to that exact same pair. Neighboring pairs however, with states and actions much alike S and A, may also benefit from the (S,A) experience. Tabular representation cannot facilitate such cross-border sharing of experience.

Fortunately a third solution exists that does: estimating $Q(S,A)$ by using linear function approximation. A set of features $f(S,A)$ can be obtained and combined linearly using weights w

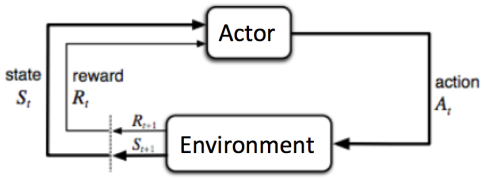


FIGURE 4: Schematic showing the cyclical life of an RL actor. Image reproduced from [13].

to approximate $Q(S,A)$:

$$Q(S,A) = w_1 f_1 + w_2 f_2 + ... + w_n f_n = w_i f_i \tag{4}$$

adopting Einstein's summation notation for a total of n weights and features. A suitable feature list serves as a summary of the current state-action pair, containing only the information significant for decision making. A unique mapping exists from a state-action pair to its feature list. Comparable state-action pairs however may result in similar or identical features. Therefore linear function approximation allows for the generalization of states, in contrast to the tabular approach. Experience gained in state-action pair (S,A) is valuable for visits to comparable state-action pairs $\hat{S},\hat{A}$ too, as long as their feature values $f(S,A)$ and $f(\hat{S},\hat{A})$ are similar. During learning, the weights w_i from Eq. 4 are updated according to the update rule in algorithm 1. We used linear function approximation in the current research as a first step and produced working mechanisms. Linear combinations are however unable to detect complex nonlinear relations between features or groups of features and the outputted value.

To cope with such non-linear relations, the linear function is replaced with a neural network (NN). The network uses 314 feature values as input layer, a single 236-node hidden layer and a single-node output layer. The value of the output node approximates $Q(S,A)$. After each time-step backpropagation is performed to determine the gradient of each weight with respect to the error. The weights are thereafter updated according to the acquired gradient information [21] using the the Adam algorithm [22]. The used NN features bias terms in both the input and hidden layer and is subject to L_1 and L_2 regularization [23].

Application to mechanism design Every episode starts with the same initial mechanism shown in Fig. 3. Subsequently the actor is allowed to choose an operator (T or D) and a link number to operate on at each time-step t. Choices are made by performing an exhaustive sweep over all possible actions A_t and ε-greedily selecting the action with the highest predicted state-action value $Q(S_t,A_t)$. Taking an action leads to a new state S_{t+1}. The actor receives a reward R_{t+1} for the new design described by S_{t+1}, calculated by the scoring module. This process repeats n times until the terminal state is reached and the episode ends. n Equals 6 by default, but can be changed as a design variable.

Feature selection

The feature set used to approximate $Q(S,A)$ should accurately describe the current state S and action A. This description should distinguishably identify unique (S,A)-pairs whilst labeling highly correlated pairs correspondingly. Whilst the former is a requirement to serve as a basis for decisions, the latter facilitates feature sets to generalize over (S,A)-pairs.

A major downside of using feature-based value-function approximation is the requirement for hand-crafted features. This requirement introduces a dependence on designer-insight in a context further dominated by artificial intelligence. This is particularly troublesome since the relation between the NN's input and output is in itself the subject of study. This relation is therefore not known a priori and hence cannot be used to determine which features are of significant value. To circumvent the issue of selection a wide variety of features have been used in this research. On the other extreme using an overly large amount of features slows down NN performance. Therefore regularization is used to perform feature selection during learning [24]. This self-selecting character contributes to the network's general applicability by adapting the features used to the design goal it is faced with. Hence, the initial features have been hand-crafted entirely independent from the adopted design goal.

A large number of features are based on graph theoretical characteristics. Included are each node's degree, betweenness, closeness, page rank, eigenvector, laplacian eigenvector, eccentricity and cluster coefficient as well as the graph's density, mean distance, efficiency, number of spanning trees, spectral gap, Fiedler value, radius, diameter, number of cycles, longest cycle length and shortest cycle length. All features in Table 1 have been scaled to fit the NN's recommended operating range between -1 and 1.

Experiment

In this section we will elaborate on the selected design objectives by means of which the efficacy of the algorithm is demonstrated. Secondly a number of design variables and algorithmic variations will be introduced, for which optimal values are established using a parameter sweep and grid search. Finally a sensitivity analysis is proposed to gain insight in the relevance of the selected features.

First design objective

The current research takes on one of the oldest and best known problems in kinematic synthesis: the straight-line problem [25]. This problem serves as a useful example since several solutions to it already exist [26, 27]. Moreover straight-lines

TABLE 1: Overview of all included features, their value type and number of values in each feature. Some feature sizes are dependent on the maximum number of nodes N or links L. The total number of features equals $||f|| = 7L + 14N + \binom{N}{2} + 17$. Since the transformation probabilities in this application are uniform, ergo the outcome of action a is always a known new state S_{t+1}, numerous features based on S_{t+1} are included. For mechanisms featuring 10 nodes, 314 features are used. Value types are boolean (B), integer (I), real(R) or miscellaneous (M)

Feature	Type	Entries
Operator selection (T or D)	B	2
Link selected as target	B	L
Existence of target link	B	1
Number of connections to target link	I	1
Number of active links in S_{t+1}	I	1
Number of active nodes in S_{t+1}	I	1
Link active in S_{t+1}	B	L
Node active in S_{t+1}	B	N
Relative link lengths in S_{t+1}	R	L
Relative node angles in S_{t+1}	R	N
Relative nodal positions in S_{t+1}	R	$2N$
Shortest path per nodal pair in S_{t+1}	R	$\binom{N}{2}$
Graph characteristics of S_{t+1}	M	$8N+11$
Node part of center in S_{t+1}	B	N
Node part of periphery in S_{t+1}	B	N
Link connected to ground in S_{t+1}	B	L
Link part of longest cycle in S_{t+1}	B	L
Link part of shortest cycle in S_{t+1}	B	L
Link part of min. spanning tree in S_{t+1}	B	L

mechanisms have served as design a challenge for computerized synthesis before [2, 28]. This design goal therefore facilitates qualitative comparison to prior art.

An objective function is implemented to determine straight-lines tracing score. The function analyses the trajectory of each node by determining the straightest section of each trajectory and fitting a minimum-surface box around the section. The aspect ratio of this box serves as a measure of the section's straightness. The final score is the product of the aspect ratio and the section's length, divided by 100 for scaling purposes.

Second design objective

The described method was developed independently from the design goal, and should therefore be able to synthesize other types of path generation mechanisms as well. To demonstrate this a second objective function has been implemented, pursuing to design figure-eights-generating mechanisms. Because of the circular nature of a figure-eight, any feasible tracing mechanism should facilitate a complete circular input motion of 2π radians. Therefore mechanisms resulting in any form of infeasible domain are immediately disregarded by the function. After disregarding such mechanisms, the function finds the remaining trajectories' principle axes and fits a scaled figure-eight on each one. An intermediary performance score is established per trajectory as the reciprocal of the normalized mean shortest distance between the trajectory points and the figure-eight. Finally the trajectories are searched for the characteristic figure-eight center crossing. The final score per trajectory is established based on the intermediate score and whether or not such a crossing is present. Fig. 8b visualizes the objective function's procedure for one trajectory. The lines between the dashed trajectory and the dotted figure-eight represent the closest euclidean distance between the trajectories. The star is plotted to mark a detected crossing.

TABLE 2: Basic settings used during parameter sweep. In each experiment these settings are adopted by all but one parameter, which is the swept parameter. The third column lists the values included in the sweep. ReLU refers to Rectified Linear Units, LReLU refers to Leaky ReLU, tanh refers to hyperbolic tangent [29], SGD to Stochastic Gradient Descent and SGDNM to Stochastic Gradient Descent with Nesterov Momentum [30]. For the property 'Hidden layer size', values relative to the input layer size are given.

Parameter	Basic value	Sweep values
Learning rate ν	$1 \cdot 10^{-1}$	$[10^{-3}, 10^{-2}, 10^{0}]$
Hidden layer style	Sigmoid	[ReLU, LReLU, tanh]
Hidden layer size	0.75	$[0.25, 0.5, 4, 16]$
L1 Regularization	$\lambda_{L1} = 0$	$[10^{-4}, 10^{-3}, 10^{-2}]$
L2 Regularization	$\lambda_{L2} = 0$	$[10^{-4}, 10^{-3}, 10^{-2}]$
Training algorithm	Adam	[SGD, SGDNM]
Primary decay rate	$\beta_1 = 0.90$,	[-]
Secondary decay rate	$\beta_2 = 0.99$	[-]
Infeasibility penalty	5	$[0, 1, 25]$
Exploration factor	100	$[30, 300]$

Variations in settings and algorithms

During the experiment numerous settings with regard to the NN have been varied. These so-called hyper-parameters are the hidden layer type, learning rate ν, number of hidden nodes, the L_1 and L_2 regularization rates and the selected training algorithm. Besides the hyper-parameters two other design variables have been varied: the decay rate for ε (exploration factor) and the level of penalty for infeasible actions and designs. The decay rates β_1 and β_2 used in the Adam algorithm are conform recommended settings [22] and kept constant throughout the experiments.

TABLE 3: Combinations of algorithm choices (TD or MC) and reward structure (accumulated or end-only) are visited in a grid search. The possible combinations are denoted as A,B, C and D.

	Temporal Difference	Monte Carlo
End-only reward	A	B
Accumulated reward	C	D

During the experiment all algorithm parameters are varied over their whole domain, whilst the remaining parameters have been kept constant according to the basic settings, both shown in Table 2. Parallel to this parameter sweep a grid-search is conducted by varying the learning algorithm and reward scheme, resulting in four different setups, see table 3. The variations A,B, C and D learn either by TD-learning, or by following an MC procedure. Moreover they use either one of two reward-schemes, accumulating rewards throughout an episode or relying solely on the episode's final reward.

Sensitivity analysis

The use of a neural network obscures the input-output relation between the features and the Q-value estimation. In order to gain an understanding of the added value of each feature sensitivity analysis is performed. However the highly nonlinear nature of NNs makes this a difficult task. Existing methods all offer different approaches with different results. Therefore four different methods are used and their results combined.

First of all the relative importance of the features is established using the weights method, introduced by Garson [31]. This method is based on partitioning the hidden-to-output weights of each hidden node into components associated with the input-to-hidden weights [32]. Using basic arithmetic operations this method leads to a relative importance score of each feature. Secondly, Lek's [33] profiling method is applied. In this method one feature value is kept fixed while all other inputs step from their minimum to their maximum value in a set amount of steps. The median of the resulting $Q(S,A)$-value is saved. This process is repeated throughout a range of values for the fixed feature, and subsequently repeated for all features. The results are used to compose a profile graph for each feature, indicating how the output varies with respect to the feature's value. Thirdly, a very basic approach is used by simply taking the mean absolute input-to-hidden weight for each feature. Finally a finite-difference sensitivity analysis is performed. During finite-difference analysis features are exposed one-by-one to a tiny increase δ and a tiny decrease $-\delta$, after which the NN output O is evaluated. The feature sensitivity is calculated as:

$$v_f = \frac{O^+ - O_-}{2\delta}, \tag{5}$$

TABLE 4: Recommended settings after performing an extensive search through 112 unique setups.

Parameter	Value
Algorithm	SARSA
Reward scheme	Reward after each time-step
Learning rate ν	$1 \cdot 10^{-2}$
Hidden layer style	Sigmoid
Hidden layer size	$\frac{3}{4}$ of input layer size
Regularization	$\lambda_{L1} = 1 \cdot 10^{-3}$, $\lambda_{L2} = 1 \cdot 10^{-3}$
Training algorithm	Adam
Decay rates	$\beta_1 = 0.90$, $\beta_2 = 0.99$
Infeasibility penalty	25
Exploration factor	30

where O^+ refers to the output after a positive perturbation and O^- after a negative one.

RESULTS

Results from parameter sweep

With a parameter sweep counting 28 design variations and a grid-search of 4 algorithm-reward combinations a total of 112 unique settings have been tested in fivefold for 5000 episodes. The results were averaged per setting in order to limit the influence of outliers. The reward distribution in the top-ten settings of both reward schemes is shown in Fig. 5. In both cases the highest median and mean is reached by the TD algorithm. The MC algorithm also demonstrates the ability to generate high-scoring mechanisms, but shows a lack of repeatability. Combining the best scoring settings with the basic setup from Table 2, recommended settings are extracted and shown in Table 4. Fig. 6 demonstrates convergence of one specific test run by means of a semilogarithmic plot of the mean-squared error propagation. The error converges towards zero with values stabilizing around $\sqrt{10^{-3}}$ after after some 2500 episodes. Convergence deteriorated for games featuring more than 6 operations per episode.

Resulting structures

The developed algorithm has synthesized a collection of functional designs, of which three well-performing individuals are shown in Fig. 7. The outer two structures (Fig. 7a and 7c) were limited to a maximum of eight nodes, whereas the structure in Fig. 7b features ten nodes. The latter design, drawing the straightest line with an aspect ratio of 1:1168, resembles Hoecken's design [34], characterized by the large structure on top of the coupler link. Interestingly, the other two structures use different approaches leading to straight lines with aspect ratios of approximately 1:410. More specifically, the structure in Fig. 7a contains a diamond

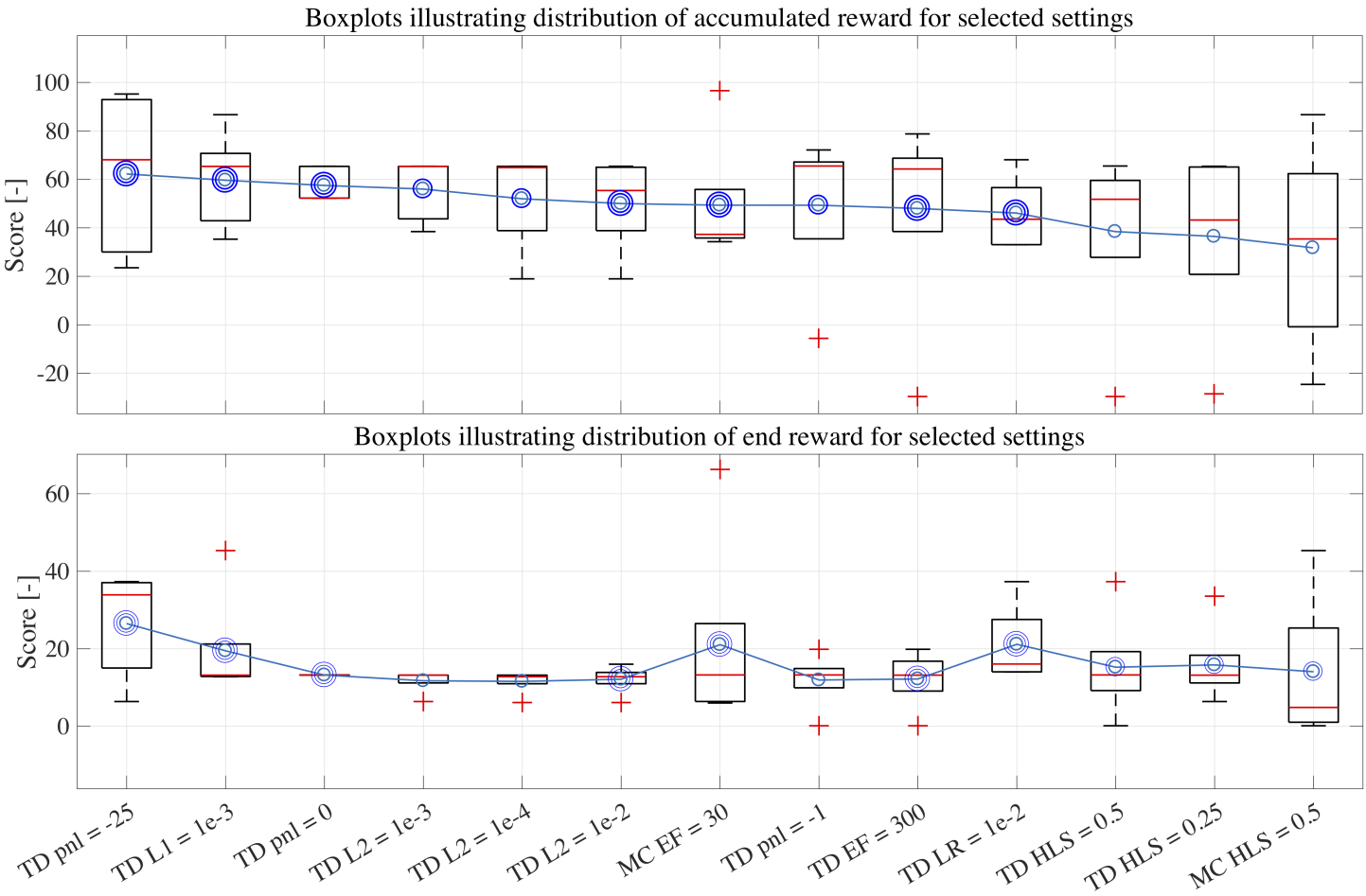


FIGURE 5: Boxplot showing the reward distribution for settings that are present in at least one of the top-tens pertaining to the highest accumulated reward (top) or end reward (bottom). The setting abbreviations are TD for Temporal Difference and MC for Monte Carlo learning. Furthermore pnl = negative reward penalty, L1 = L_1 regularization parameter, L2 = L_2 regularization parameter, EF = Exploration factor, LR = Learning rate ν, HLS = hidden layer size. The distributions are based on experiments in fivefold. Results are marked as outliers when deviating more than 2σ from the mean, illustrated by the red plus-signs. The mean values are represented by the circles. Triple circle markers indicate settings present in both top-tens. Double circle markers represent top-ten settings unique to the figure's corresponding reward scheme. Single circle markers represent settings scoring below top-ten.

shaped subset, of which the top node passes through a straight section. While following the radial motion of the input link, the diamond folds in and out, counteracting movement perpendicular to the straight section. A similar folding movement was used by Peaucellier in 1873 [26] in his perfect straight-line mechanism.

Benchmarking

The straight-line objective function in the presented method is similar to the one adopted by Lipson [2]. This allows for a quantitative comparison between the current RL approach and Lipson's evolutionary algorithm (EA).

However it should be noted that at least three major discrepancies exist between the two methods, making the comparison difficult to interpret. First of all the positioning of new nodes differs between the two methods, as depicted in Fig. 2. The current node-placement method does not feature local coordinates, which limits the solution space to only a subset compared to [2]. Secondly the current method evaluates a trajectory on both straightness and length, whereas Lipson's solution adheres to the aspect ratio only. Thirdly minuscule inaccuracies in the calculated trajectories, through numeric simulation, may disturb an otherwise high aspect ratio. To test this hypothesis the current numeric simulator was used to determine the aspect ratio of a Peaucellier machine's trajectory, which is known to be per-

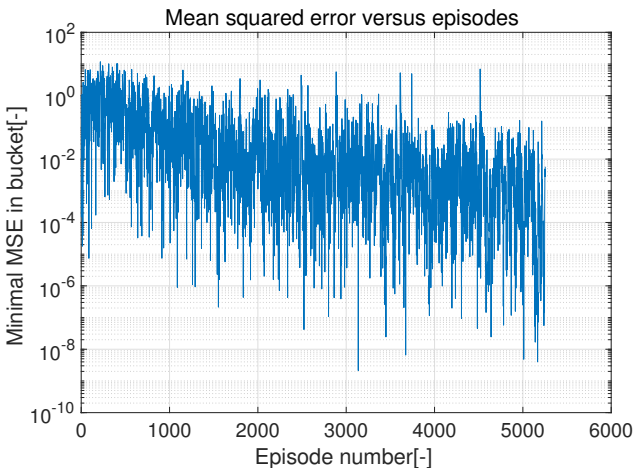


FIGURE 6: Semilogarithmic plot showing the propagation of the mean squared error through a series of episodes. Random events, caused by exploration, result in high error spikes. Therefore, the minimum errors per bucket of 20 error-values are shown in this figure.

fectly straight. Upon analyzing the trajectory an aspect ratio of 1:1947 was obtained. This indicates the simulator's accuracy is insufficient to measure aspect ratios in this extreme region. For reference, Lipson's evaluation method found a $1:10^5$ ratio for the same machine [35].

That being said ratios can be compared. Lipson [2] describes most of the resulting structures exceed aspect ratios of 1:1000, with outliers as high as 1:28340. The results in Fig. 7 show aspect ratios of respectively 1:415, 1:1168 and 1:406. Therefore the straightness in the current results do not challenge the levels of straightness achieved by Lipson. The required computational time for a complete learning procedure is in both cases in the order of 10 hours. For the results in this paper a 2.7 GHz Intel Core i5 was used, whereas Lipson used a 1.5 GHz Pentium IV.

Sensitivity analysis

The results from the four sensitivity analyses vary. There are however five features that belong to the ten most sensitive features in all analyses. These are shown, in arbitrary order, in Table 5.

The order of merit between these features varies over the different analysis methods and is therefore not included. Nonetheless, the table yields informative results. The strong positive relation found between Q and the existence of the target link is rather trivial, since actions on non-existing links lead directly to an infeasibility penalty. The next two however are not so trivial: operations on link 6 lead show a positive sensitivity, whereas operations on link 4 are negatively related to the expected reward. However, this is not supported by the results: successful mechanisms shown in Fig. 7a and Fig. 7b are the result of operations on both link 4 and 6.

TABLE 5: Most sensitive features with respect to Q.

Feature	Relation to Q
Existence of target link	Positive
Link 6 is selected as target	Positive
Link 4 is selected as target	Negative
Link 7 is part of longest cycle	Positive
Relative vertical position of node 4	Negative

Results of second design objective

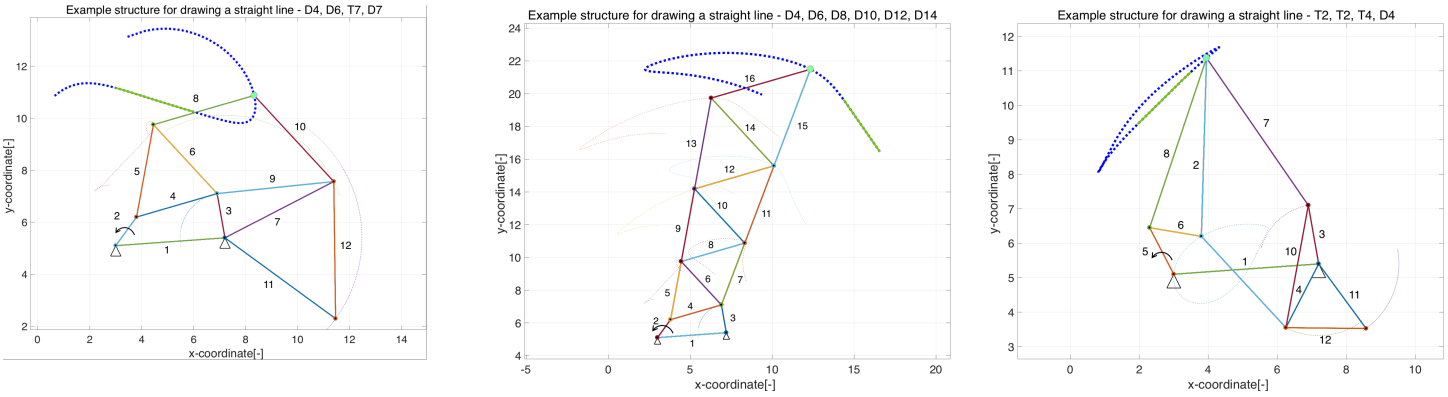
The learning algorithm has been tested using a second design objective, utilizing the same settings and features used to synthesize straight-line mechanisms. The resulting design is shown in Fig. 8a. The depicted mechanism clearly traces a figure-eight, albeit curved. Even though the imposed restrictions do not allow for the creation of a perfect figure-eight trajectory, the results demonstrate the algorithm's capability of adapting to other design goals.

DISCUSSION

The results presented in this paper prove that RL can be applied to synthesize straight-line mechanisms. Therefore this demonstration broadens the mechanical system design spectrum of tools and gives future researchers an extra angle to consider whilst choosing a fitting synthesis technique. The current research may serve as a stimulant for the engineering world to start adopting machine learning into its range of design tools.

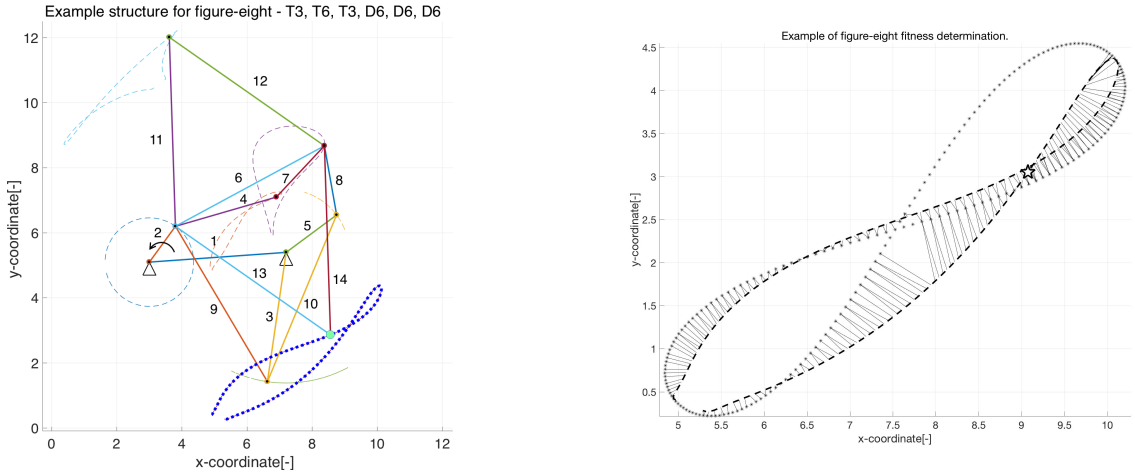
Zooming out, one sees how this paper opens the door to a new area of application for the heavily researched field of RL. DeepMind's David Silver [36] recently said: "Now we can start to tackle some of the most challenging and impactful problems of humanity", after having presented a breakthrough in RL. Such an announcement indicates a movement in the RL community towards the application of its skills to other research fields. By demonstrating the efficacy of RL in mechanical engineering this research removes a barrier for scientists like Silver to apply their craft to mechanical system design.

Extensions on the current results could be made by removing the imposed restrictions on Hod Lipsons's [2] T and D operators. A first step would be to extend the actor's freedom in choosing the location of new nodes. A paramount improvement however could be made by using a learning algorithm capable of coping with a continuous action space. Such algorithms (policy gradient [37], actor-critic models [18, 38]) do not use an `argmax` operator to perform greedy action selection, but instead estimate a stochastic reward distribution over the action space and select an action accordingly, decoupling computational time from the size of the



(a) (b) (c)

FIGURE 7: Three structures as created by the algorithm. The thick, dark blue dashed lines represent the trajectories featuring a straight section, highlighted by a full green line. The large green nodes represent each structure's straight-line tracing node. The aspect ratios of the straight sections are (a) 1:415, (b) 1:1168 and (c) 1:406.



(a) (b)

FIGURE 8: Example of the application of the proposed algorithm adopting a figure-eight design goal. Figure (a) shows the resulting structure, in which the green node (lower-right corner) traces a blue thick dashed figure-eight pattern. Figure (b) shows how the trajectory is compared to a true figure-eight. The colored lines display the shortest distance between each trajectory point and the true figure-eight. In this case, the mean euclidean distance equals 0.203. The green star indicates a detected crossing.

action space. The resulting algorithm could search a continuous design space from which it may obtain designs that achieve more accurate performance.

Secondly the value function approximator could be improved by further researching the ideal set of features. The presented result from the sensitivity analyses has introduced only limited insight in the inner workings of the network. The effect of individually important features may have been outweighed by a combined effort from less important features. Also, Q might be very sensitive to certain groups of features moving together. Unfortunately both of these phenomena cannot be detected by the applied analysis methods. As a first step towards feature selection however, the presented methods can be used to remove obsolete features, increasing the algorithm's efficiency.

Besides the issues concerning feature selection, the neural network's layer depth may be extended. Such a deep neural network would be able to detect more complex relations between sets of features and the expected reward, leading to increased algorithm performance. Finally some post-processing steps may be developed to clean up the algorithm's results. For example, post-processing could remove obsolete triangles or adapt dimensions to improve aesthetics and producability of the designs. The

resulting software may be directly applicable to the design of mechanisms for specific kinematic goals, whether it be tracing trajectories, amplifying motion or other kinematic challenges.

Additionally, quantitative benchmarking showed room for improvement on the part of the design accuracy of the proposed algorithm. However the imposed restriction on the algorithmic design freedom impairs the fairness of comparison, as did small inaccuracies of the obtained kinematic simulation results. By adopting smaller tolerances in the kinematic modeling procedure these inaccuracies can be minimized, moving performance towards Lipson's.

Finally, the results from the sensitivity analysis were not conclusive. One can hypothesize that these individual sensitivities should be viewed in context of the neural network: the effect of one feature, albeit a highly sensitive one, may be outweighed by a group of other features. Also the network's output may be much more sensitive to simultaneous shifts in certain combinations of features. Analyzing individual features therefore introduces only limited insight in the Neural Network's inner workings.

CONCLUSION

The current research demonstrates the added value of reinforcement learning in mechanism design, which has thus far been uncultivated ground. In this paper a method is presented by which kinematic synthesis can be molded into a game-like process compatible with reinforcement learning algorithms.

We have explored the use of value-function approximation by a neural network to predict the performance of a kinematic mechanism. An extensive parameter search resulted in a list of recommended algorithm and hyper-parameters settings.

A series of straight-line mechanisms has been successfully and independently synthesized by the developed algorithm. Although no exact solutions have been found, the resulting designs are unmistakably straight-line mechanisms and show similarities to Peaucellier's [26] and Hoecken's [34] mechanisms.

In addition we demonstrated the algorithm's ability to synthesize a mechanism for a figure-eight path-generation objective. The successful handling of a second design goal suggests the algorithm's potential as a general solution for a variety of kinematic synthesis challenges.

In future work the applied SARSA algorithm can be upgraded to an actor-critic or policy gradient model, effectively alleviating the necessity of the current restrictive measures on the nodal placement and thereby increasing the algorithms design freedom. Combined with further work on feature selection and neural network architecture, reinforcement learning should be regarded as a promising means for the synthesis of mechanical systems.

REFERENCES

[1] Yin, L., and Ananthasuresh, G. K., 2003. "Design of Distributed Compliant Mechanisms". *Mechanics Based Design of Structures and Machines,* ***31***(2), pp. 151–179.

[2] Lipson, H., 2008. "Evolutionary synthesis of kinematic mechanisms". *AI EDAM,* ***22***(03), aug, pp. 195–205.

[3] Zhou, H., and Ting, K.-L., 2005. "Topological Synthesis of Compliant Mechanisms Using Spanning Tree Theory". *Journal of Mechanical Design,* ***127***(4), p. 753.

[4] Saxena, A., 2005. "Synthesis of Compliant Mechanisms for Path Generation using Genetic Algorithm". *Journal of Mechanical Design,* ***127***(May 2010), p. 745.

[5] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D., 2017. "Mastering the game of Go without human knowledge". *Nature,* ***550***(7676), pp. 354–359.

[6] Okdinawati, L., Simatupang, T. M., and Sunitiyoso, Y., 2017. "Multi-agent reinforcement learning for collaborative transportation management (ctm)". In *Agent-Based Approaches in Economics and Social Complex Systems IX*. Springer, pp. 123–136.

[7] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A., 2017. "Target-driven visual navigation in indoor scenes using deep reinforcement learning". In Robotics and Automation (ICRA), 2017 IEEE International Conference on, IEEE, pp. 3357–3364.

[8] Kaelbling, L. P., Littman, M. L., and Moore, A. W., 1996. "Reinforcement learning: A survey". *Journal of Artificial Intelligence Research,* ***4***, pp. 237–285.

[9] Bernardoni, P., Bidard, C., and Drt, C. E. A., 2004. "A new compliant mechanism design methodology based on flexible building blocks". pp. 244–254.

[10] Kawamoto, A., Bendsøe, M. P., and Sigmund, O., 2004. "Planar articulated mechanism design by graph theoretical enumeration". *Structural and Multidisciplinary Optimization,* ***27***(4), pp. 295–299.

[11] Avilés, R., Hernández, A., Amezua, E., and Altuzarra, O., 2008. "Kinematic analysis of linkages based in finite elements and the geometric stiffness matrix". *Mechanism and Machine Theory,* ***43***(8), pp. 964–983.

[12] Battiti, R., 1992. "First-and second-order methods for learning: between steepest descent and newtons method". *Neural computation,* ***4***(2), pp. 141–166.

[13] Sutton, R. S., and Barto, A. G., 2012. *Reinforcement Learn: An Introduction*, 2 - draft ed. The MIT Press.

[14] Melo, F. S., and Ribeiro, M. I., 2007. "Q-Learning with Linear Function Approximation". In *Learning Theory*, Vol. 1. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 308–322.

[15] Markov, A. A., 1953. "The theory of algorithms".

[16] Banach, S., 1922. "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales". *Fund. Math,* ***3***(1), pp. 133–181.

[17] Szepesvári, C., 2010. "Algorithms for Reinforcement Learning". *Synthesis Lectures on Artificial Intelligence and Machine Learning,* ***4***(1), pp. 1–103.

[18] van Hasselt, H., and Wiering, M. A., 2007. "Reinforcement Learning in Continuous Action Spaces". In 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, no. Adprl, IEEE, pp. 272–279.

[19] Perez-Uribe, A., and Sanchez, E., 2009. "Blackjack as a test bed for learning strategies in neural networks". In 1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227), Vol. 3, IEEE, pp. 2022–2027.

[20] Abul, O., Polat, F., and Alhajj, R., 2000. "Multiagent reinforcement learning using function approximation". *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* ***30***(4), pp. 485–497.

[21] Hecht-Nielsen, R., et al., 1988. "Theory of the backpropagation neural network.". *Neural Networks,* ***1***(Supplement-1), pp. 445–448.

[22] Kingma, D. P., and Ba, J., 2015. "Adam: A Method for Stochastic Optimization". In 3rd International Conference for Learning Representations.

[23] Ng, A. Y., 2004. "Feature selection, L1 vs. L2 regularization, and rotational invariance". *Twenty-first international conference on Machine learning - ICML '04*, p. 78.

[24] Kolter, J. Z., and Ng, A. Y., 2009. "Regularization and Feature Selection in Least-Squares Temporal Difference Learning". In Proceedings of the 26th annual international conference on machine learning., ACM, pp. 521—-528.

[25] Tarabarin, V., Tarabarina, Z., and Chirkina, D., 2012. "Designing, Analysis and Computer Modeling of Straight-Line Mechanisms". In *Explorations in the History of Machines and Mechanisms*. pp. 551–563.

[26] Peaucellier, C., 1873. "Note sur une question de geometrie de compas". *Nouv. Ann. der Math,* ***12***, pp. 71–81.

[27] Dijksman, E. A., 1972. *Approximate straight-line mechanisms through four-bar linkages*. Editions de l'Académie de la Republique Socialiste de Roumanie.

[28] Kuppens, P., 2016. "Automated Robot Design With Artificial Evolution". Msc. thesis, Delft University of Technology.

[29] Fei-Fei Li, Justin Johnson, S. Y., 2017. Lecture 6: Training neural networks, part 1 - stanford university cs231n. Lecture slides. Available on http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf , visited on November 22nd 2017, Stanford University.

[30] Nesterov, Y., 1983. "A method of solving a convex programming problem with convergence rate o (1/k2)". In Soviet Mathematics Doklady, Vol. 27, pp. 372–376.

[31] Garson, D. G., 1991. "Interpreting neural network connection weights". *AI Expert,* ***6***(7), pp. 47–51.

[32] Gevrey, M., Dimopoulos, I., and Lek, S., 2003. "Review and comparison of methods to study the contribution of variables in artificial neural network models". *Ecological modelling,* ***160***(3), pp. 249–264.

[33] Lek, S., Belaud, A., Baran, P., Dimopoulos, I., and Delacoste, M., 1996. "Role of some environmental variables in trout abundance models using neural networks". *Aquatic Living Resources,* ***9***(1), pp. 23–29.

[34] Lu, S., Zlatanov, D., Ding, X., and Molfino, R., 2014. "A new family of deployable mechanisms based on the hoekens linkage". *Mechanism and Machine Theory,* ***73***, pp. 130–153.

[35] Lipson, H., 2006. "A Relaxation Method for Simulating the Kinematics of Compound Nonlinear Mechanisms". *Journal of Mechanical Design,* ***128***(4), p. 719.

[36] DeepMind YouTube Channel, 2017. Alphago zero: Discovering new knowledge.

[37] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M., 2014. "Deterministic Policy Gradient Algorithms". *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395.

[38] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., 2015. "Continuous control with deep reinforcement learning".