

# Hybrid Genetic Programming and Deep Reinforcement Learning for Low-Complexity Robot Arm Trajectory Planning

Quentin Vacher<sup>a</sup>, Nicolas Beuve<sup>b</sup>, Paul Allaire<sup>c</sup>, Thibaut Marty<sup>d</sup>, Mickaël Dardaillon<sup>e</sup> and Karol Desnos<sup>f</sup>

Univ Rennes, INSA Rennes, CNRS, IETR – UMR 6164, F-35000 Rennes, France  
{firstname.lastname}@insa-rennes.fr

Keywords: Genetic Programming, Reinforcement Learning, Inverse Kinematics.

Abstract: Robot arm control is a technological challenge where an algorithm needs to learn a deep understanding of spatial navigation. In particular, spatial navigation requires learning the relationship between the motor joint angular positions and the Cartesian coordinates of the robot. Trajectory planning is an even more complex challenge, where the algorithm must create a trajectory between two coordinates that does not cause a collision. State-of-the-art algorithms capable of solving trajectory planning are based on deep Reinforcement Learning (RL). These algorithms achieve high accuracy but suffer from high computational complexity. This paper proposes to use a genetic RL algorithm, the Tangled Program Graphs (TPGs), to solve trajectory planning. Using a genetic process, the TPGs generate a graph of programs with low inference complexity. On a first trajectory planning problem, the algorithm used achieves performance close to the state-of-the-art, but with a 100 less execution time and a 20× smaller model size. On a second and more difficult problem, the TPGs are not able to learn with efficiency. We propose a hybrid solution that mixes the TPGs and a state-of-the-art deep RL algorithm, the Soft Actor-Critic (SAC). This solution achieves better performance than the state-of-the-art for both problems, with 6 to 20 times faster execution times.

## 1 INTRODUCTION

Robot control presents a formidable technological challenge in today's landscape. At its core, this challenge entails the development of algorithms that possess a deep understanding of spatial relationships. This understanding is crucial for orchestrating the coordinated movements of the various motors of the robot. As robots become increasingly ubiquitous across industries, there is a pressing need to optimize the computational costs associated with controlling them. This optimization ensures efficiency without compromising performance, a balancing act essential for widespread adoption and practical implementation.

Some recent techniques, such as Deep Learning, can efficiently control robot models. Deep Learning

solutions (Giorelli et al., 2015) effectively predict the motor joint values of a robot to reach a desired position. However, these solutions struggle to avoid collisions during trajectory planning.

Deep Reinforcement Learning (RL) methods can efficiently learn how to avoid collisions (Zhong et al., 2021). This class of methods is well-suited for trajectory planning, but suffers from a high computational complexity.

We propose to use a Genetic Programming (GP) algorithm, the Tangled Program Graph (TPG), to do trajectory planning. This algorithm uses GP evolution to generate a graph designed for a specific task with really low complexity. With only some tuning of the training hyperparameters, we show that the TPGs achieve state-of-the-art RL performance on a Inverse Kinematics (IK) task, while being 100 times less computationally expensive and having a model size 20 times smaller. However, on a more difficult trajectory planning task with more diversity, the TPGs alone are not able to reach equivalent accuracy.

To take advantage of both the low complexity of the TPGs and the high accuracy of a deep RL algorithm, the Soft Actor-Critic (SAC), a hybrid solution

<sup>a</sup> <https://orcid.org/0009-0001-9568-7196>

<sup>b</sup> <https://orcid.org/0000-0002-1371-4016>

<sup>c</sup> <https://orcid.org/0009-0005-9325-0643>

<sup>d</sup> <https://orcid.org/0000-0001-7035-8727>

<sup>e</sup> <https://orcid.org/0000-0001-6862-2090>

<sup>f</sup> <https://orcid.org/0000-0003-1527-9668>

is proposed. This hybrid solution achieves better results than the state-of-the-art deep RL on trajectory planning with collisions, while being 6 to 20 times faster in inference. Because the hybrid solution combines both algorithms, the inference time of the TPGs alone is still 4 to 17 times faster than the hybrid solution at inference. The model size is the combination of both the deep RL and the TPG models, so it is slightly larger than the state-of-the-art.

The various algorithms and principles used in this paper are described in Section 2. The experimental setup, which describes the environment of the robot and the implementation of the TPGs used, is presented in Section 3. Section 4 presents the performance of the TPGs and the used deep RL algorithm, the SAC (Haarnoja et al., 2018). Section 5 describes the hybrid solution that combines the TPGs and the SAC and compares it with the state-of-the-art. Finally, a discussion is made in Section 6 and Section 7 concludes this paper.

## 2 RELATED WORKS AND BACKGROUND

This section introduces the Inverse Kinematics (IK) problem. Section 2.1 presents the problem and the most commonly used existing solutions. Then Section 2.2 describes a branch of IK which is the main focus of this paper: the trajectory planning. Section 2.3 introduces the RL field and the reason why deep RL solutions are used to solve trajectory planning problems. Finally, Section 2.4 introduces the TPGs, describing the principles and applications of this GP algorithm.

### 2.1 Inverse Kinematics

Inverse Kinematics (IK) is a fundamental problem in robotics that deals with determining joint configurations for a robot manipulator. The goal is to place the end-effector in a given position and orientation, where the end-effector is the hand of the robot. The spatial coordinates describing the position and orientation of the end-effector are defined as  $x \in \mathbb{R}^n$ , where  $n$  is the number of dimensions. The motor angular coordinates are defined as  $q \in \mathbb{R}^m$ , where  $m$  is the number of joints, as illustrated in Figure 1a. The forward kinematics problem is equivalent to finding the function  $f(\cdot)$  that satisfies the following equation:

$$\mathbf{x} = f(\mathbf{q}) \quad (1)$$

The function  $f(\cdot)$  is based on trigonometric functions, has a complexity of  $O(n)$  and a unique solu-

tion. The IK problem consists in finding the function  $f^{-1}(\cdot)$  that calculates the joint coordinates of the robot  $q$  from the position and orientation of the end-effector  $x$ . As illustrated in Figure 1b, this function can have multiple solutions (Ho and King, 2022) if the number of joint dimensions  $m$  is greater than the number of space dimensions  $n$ . Depending on the implemented robot control strategy, the goal of IK may be to find all solutions or the one that best satisfies certain criteria. Criteria can be to find the position that requires the minimum motor movement or the position that maximizes the stability of the robot (Phaniteja et al., 2017). There exist several solutions to solve the IK problems, which can be divided into three categories: analytical, numerical and learning solutions.

- Analytical solutions (Sciavicco and Siciliano, 2012) address the kinematic equations directly. They are effective for problems with few Degrees of Freedom (DOF) and have a low computational cost while predicting the exact solution. However, for larger problems with many joints to control, analytical methods may never find a solution.
- Numerical solutions, such as the Jacobian transpose (Wolovich and Elliott, 1984) or the damped least squares method (Wampler, 1986), solve the IK problem, by iteratively computing an approximation to a solution of the function  $f^{-1}(\cdot)$ . Because of this iterative nature, they have a higher computational cost than analytic solutions, and the solution may depend on the first approximation used to initialize the iterations.
- The emergence of deep learning in recent years has brought new solutions to the IK problem. Deep learning is known to be very efficient at solving high-dimensional problems, so it is naturally well suited to the IK problem. A standard feedforward neural network outperforms the accuracy of the Jacobian transpose methods (Giorelli et al., 2015) while computing the result 100 times faster.

### 2.2 Trajectory Planning

Trajectory planning is a branch of IK where the goal is to move the end-effector from a coordinate  $x_{src}$  to a target coordinate  $x_{dst}$  and to control the path taken along the way to avoid collisions or unavailable positions.

Deep learning or numerical methods usually compute trajectories by defining intermediate coordinates, creating a trajectory, and computing the position of the motor joints coordinate by coordinate (Duka,

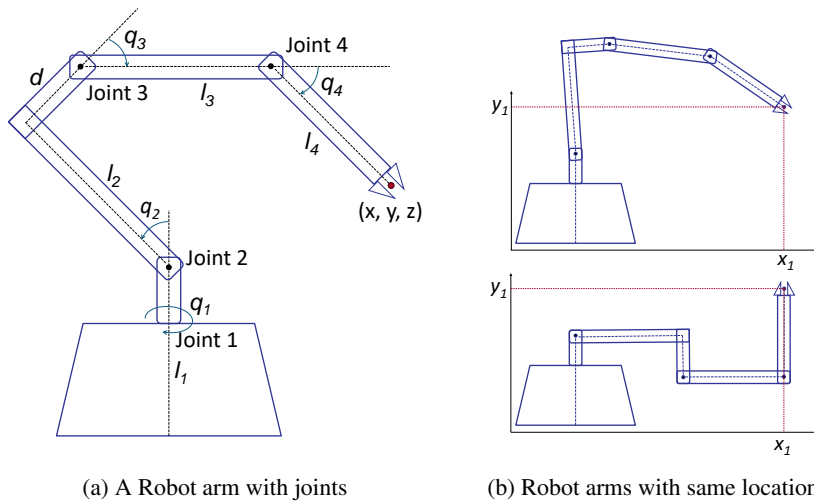


Figure 1: The 4-DOF Robot arm used in the experiment. (a) The arm with the motor joints (black dots) and the end-effector coordinates (red dot). (b) A case where two coordinates reach the same end-effector coordinates.

2014). This results in weak collision avoidance and leaves the possibility that two intermediate coordinates will result in completely different motor positions. If the method can compute all the mappings of the function  $f^{-1}(\cdot)$  (Ho and King, 2022), then taking the closest mapping to the previous coordinates can solve the motor correlation. However, collision avoidance in trajectory planning is much harder to solve with these methods.

### 2.3 Reinforcement Learning Solutions for Trajectory Planning

RL is a branch of machine learning in which an agent interacts with an environment by taking actions that yield rewards (Sutton and Barto, 2018). The objective of the agent is to learn a policy, that is, a mapping from states to actions, that maximizes the cumulative reward over time.

RL is a good solution for trajectory planning because the agent can create optimized trajectories based on a reward defined by the learning environment. The reward is usually the distance between the end-effector and the target, but a penalty can be added to encourage the agent to avoid collisions (Phaniteja et al., 2017).

Some RL algorithms using deep learning have shown great efficiency in robot control (Ibarz et al., 2021), such as the Soft Actor-Critic (SAC) (Haarnoja et al., 2018). However, these solutions use deep learning networks that require too much computation for a deployment on tightly resource-constrained embedded systems. The SAC model from (Haarnoja et al., 2019) uses 2 layers of 256 units each, leading to at

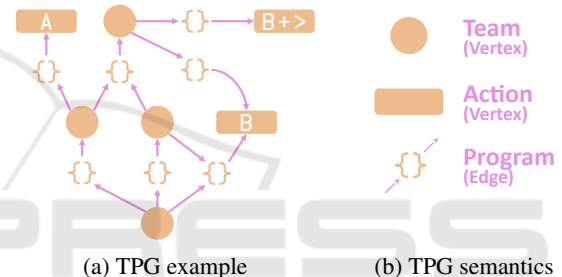


Figure 2: Semantics of the TPGs.

least 65k parameters, resulting in models of about 300 kB.

### 2.4 Tangled Program Graphs

Today, Deep RL methods are the most widely used algorithms for solving trajectory planning in the IK problem. However, other RL techniques such as the TPG could be used to solve this problem. TPG is an algorithm introduced by Kelly and Heywood (Kelly and Heywood, 2017) based on GP principles that has been shown to be efficient for RL problems at very low computational cost.

#### 2.4.1 The TPG Model

A TPG, depicted in Figure 2, consists of three main entities that form a directed graph: the programs, the teams, and the actions. The actions and teams are the vertices of the graphs, while the programs are the edges between a source and a target team or action. Teams are internal vertices, while actions are the leaves of the graph. Each team has several outgoing edges.

### 2.4.2 The Program

A program is a list of instructions constructed during the genetic evolution training process. A set of instructions is defined by the programmer prior to the training process. The program maps the state vector with the different instructions to produce a bid. During the inference process, a team follows the path of the program with the highest bid and reaches the corresponding vertex. The instructions are crucial for the selection of actions and can be adapted for each type of problem (Desnos et al., 2021).

### 2.4.3 The Evolution Process

The structure of the graph evolves dynamically. A fixed number of roots are created in each generation by duplicating and mutating roots that have survived the natural selection process of the previous generation. The mutation of a root can change either the destination of the edges of the root, or the instructions of the programs on the edges.

The TPGs are able to solve RL problems, such as the Atari games (Kelly and Heywood, 2017), with low inference complexity. The evolutionary process used to train the TPGs produces RL agents with an adaptive computational complexity that depends on the difficulty of the problem. Unlike the TPGs, deep RL methods require a specific model size defined by a data scientist, which is about 100 times larger than a TPG and takes about 100 to 1000 times longer to run than a TPG (Kelly, 2018) for equivalent accuracy. These characteristics make the TPGs a good candidate for the trajectory planning problem.

## 3 EXPERIMENTAL SETUP

This section presents the experimental setup used to evaluate the TPGs, the SAC, and the hybrid solution on trajectory planning tasks. Section 3.1 describes the robot arm model used. Section 3.2 explains what the environment is for the TPGs and its different components, then Section 3.3 describes some differences between the environment used by the SAC and the TPGs.

To ensure reproducibility of the results presented, all library code, pre-trained TPGs, execution traces and analysis code are made available as open-source artifacts (Vacher et al., 2024)

### 3.1 4-DOF Robot Arm

The KIT-WIDOWX-ARM-COMP robot arm used in the experiment has 4-DOF as represented on Fig-

ure 1a. It is a three-link arm placed on a base  $l_1$  with rotational joints. The joints are limited to range  $[-\frac{\pi}{2}, \frac{\pi}{2}]$ , except for the base, which can do a full rotation, but cannot go from  $-\pi$  to  $\pi$ . The RL algorithms are trained on a simulator built with Orocos library and a model characterizing of the valid positions was built using physical constraints of the actual arm.

The model of the robot arm used in the experiment has 4-DOF, Figure 1a shows a representation of the arm. The arm used is the KIT-WIDOWX-ARM-COMP. It has six servomotors but two of them control the hand which is not considered in this study. It is a three-link arm, as shown in Figure 1a, placed on a base  $l_1$  with rotational joints and different sizes of joints:

$$\begin{aligned} l_1 &= 125 \text{ mm}; & l_2 &= 142 \text{ mm}; & l_3 &= 142 \text{ mm}; \\ l_4 &= 155 \text{ mm}; & d &= 49 \text{ mm} \end{aligned}$$

Motor angular possibilities are limited within the following ranges (note that  $q_1$  can not go from  $-\pi$  to  $\pi$ ):

$$\begin{aligned} q_1 &\in [-\pi, \pi]; & q_2 &\in [-\frac{\pi}{2}, \frac{\pi}{2}]; \\ q_3 &\in [-\frac{\pi}{2}, \frac{\pi}{2}]; & q_4 &\in [-\frac{\pi}{2}, \frac{\pi}{2}] \end{aligned}$$

### 3.2 The Environment for the TPGs

**The Environment State.** The state of the environment as given to the RL agent is composed of 13 values divided into four vectors :

1. The four motor positions;
2. The position of the hand  $P_h = (x_h, y_h, z_h)$ ;
3. The position of the target  $P_t = (x_t, y_t, z_t)$ ;
4. The difference between the hand and the target  $P_d = (x_h - x_t, y_h - y_t, z_h - z_t)$ ;

Vector four may seem irrelevant because it is the difference between vectors two and three, but it has been empirically found to significantly speed up the training process for both the TPG and SAC algorithms.

**The Action Space.** TPGs can only perform a single discrete action per inference. The action rotates one of the four motors either clockwise or counter-clockwise by 3 motor steps, where a step is approximately 0.0046 rad or 0.26°. With a smaller motor step size, the results can be more accurate, but require more actions to learn because the number of actions needed to reach a target is higher. Hence, there are eight available actions: that is 2 actions per motor, plus a ninth action which is a no-action that causes self-termination.

**Objective of the Learning.** The robot arm must move from coordinates  $(x_{src}, y_{src}, z_{src})$  to

$(x_{dst}, y_{dst}, z_{dst})$ . A position is considered reached when the distance between the end-effector and the target is less than three millimeters. This value of three millimeters is arbitrarily chosen. Two different trajectory planning problems are considered, the first being simpler than the second:

- **Fixed Start (FS)**. Go from a fixed starting position to a random destination.
- **Random Start (RS)**. Go from a random starting position to a random destination.

For the Fixed Start (FS) problem, all targets can be reached in a maximum of 1707 actions. The Random Start (RS) problem doubles this to 3414. These values are the maximum number of actions during an episode to reach the target.

At each generation, the roots are tested on 100 targets, making 100 episodes per generation. A validation is performed at each generation with the five best roots of the training with 100 validation targets. Two unique sets of validation targets, one for each problem, are used for all the results presented in this paper. The targets are sampled uniformly in the reachable 3D space.

**The Score.** Contrary to classic Reinforcement Learning (RL) algorithms, the TPGs do not use an instant reward and the Bellman equation (Bellman, 1957) to learn, but only a score at the end of each evaluation. The score  $S$ , as specified in Equation (2), is the distance in millimeters to the target multiplied by  $-1$  if the distance  $d$  between the target and the hand is above 3 millimeters. If the distance is lower, then the objective is reached and the score becomes positive to encourage to take fewer actions to reach the target.

$$S = \begin{cases} -1 \times distance & \text{if } d \geq 3 \\ (nbMaxActions - nbActions) \times 0.01 & \text{if } d < 3 \end{cases} \quad (2)$$

Since 100 episodes are evaluated for each root per generation, the score of a root is the average score of the 100 episodes.

Invalid positions occur when any part of the robot is below the bottom of the arm, assuming the robot is attached to a table. If a collision or an invalid position is detected, the action is aborted, and the environment state remains unchanged. For the TPGs, a root will always choose the same action with the same environment state. Consequently, the same action will be selected repeatedly, causing the arm to stop. As a result, the score will reflect the current negative distance, which is unfavorable compared to a successful outcome. The selection will naturally avoid collisions and invalid positions.

### 3.3 The Environment Variations for the SAC

The environment used by the SAC is mainly the same as the environment for the TPGs described in the previous section.

Using the same reward as the TPGs would not be fair for either the SAC or the TPGs, because both algorithms learn differently. The TPGs learn with a final score at the end of a generation while the SAC learns with a reward at each action step. The used reward  $r$  is described by Equation 3, where  $d$  is the distance in millimeters between the hand and the target.

$$r = \begin{cases} -0.1 \times d & \text{if } d \geq 3 \text{ \& no collision} \\ -0.1 \times d - 10 & \text{if } d \geq 3 \text{ \& collision} \\ 100 & \text{if } d < 3 \end{cases} \quad (3)$$

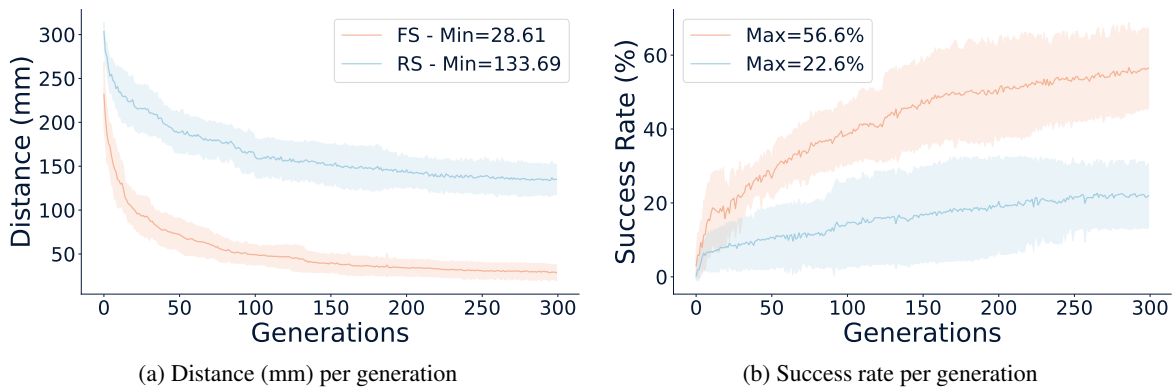
Unlike the TPGs, the SAC can perform multiple actions at once, thus the SAC has four possible actions, each of which rotates one of the four motors either clockwise or counterclockwise by 3 motor steps. Because the SAC uses multiple actions, it is limited to 1000 actions per episode instead of 1707. The limit could be higher because the reward of the SAC does not depend on the maximum number of actions, but some earlier tests showed that it does not affect the learning of the SAC.

For the TPGs, an episode ends when a collision occurs, as explained in the description of the score in section 3.2, but this also corresponds to the goal of learning: to avoid collisions. The training of the SAC was initially done with the same behavior, where a collision would end the episode. The problem is that this creates episodes with different numbers of actions. Since the goal of RL is to maximize the sum of rewards over an episode, this created instability in the training.

To avoid this problem, during the training of the SAC, a collision does not end the episode but accumulates negative rewards more quickly. However, for comparison with the TPGs in inference, a collision ends the current episode. The SAC is referred as SAC No Collision on inference (NC<sub>i</sub>).

## 4 RESULTS OF TPGs AND SACs

Section 4.1 presents the results of the TPGs, then the results of the SAC are shown in Section 4.2. Finally, in Section 4.3, a comparison of both algorithms is done.



(a) Distance (mm) per generation

(b) Success rate per generation

Figure 3: Results of the TPGs for the FS problem, with fixed starting positions, and the RS problem, with random starting positions. The curves show the mean of the ten seeds  $\pm$  the standard deviation in the shaded areas. (a) Shows the distance results. (b) Shows the percentage of success.

#### 4.1 Results of the TPGs

The first experiments with the TPGs on trajectory planning are done for the FS problem, where the initial position of the arm is fixed between each episode. Out of the box, training the TPGs with the training hyperparameters proposed in Kelly's work only achieves a mean distance to the target of 94.5 millimeters over 5 seeds after 150 generations.

An empirical study has been done to improve the results by tuning the hyperparameters. The parameters studied are the number of roots at each generation, the number of surviving roots at each generation and the number of initial roots. The number of roots did not change, but the number of surviving roots and initial roots have changed from 180 and 180 to 20 and 7200, respectively. With these parameters, the TPGs achieve an average distance to target of 38.8 millimeters over 10 seeds after 150 generations. The training was then extended to 300 generations to improve the results. The TPGs achieve an average distance to target of 28.61 mm and a success rate of 57% for the FS problem.

A training with the same configuration was done for the RS problem, where the initial position is random. The TPGs achieve a distance to target of 133.69 mm and a success rate of 22.6% for the RS problem. Figure 3 shows the training results of the TPGs for both objectives.

#### 4.2 Results of the SAC

The SAC is trained on 5 seeds using the parameters in Table 1. One hundred episodes are done for the training before a validation step of 100 episodes. To evaluate a low complexity SAC, a training is done with 16 hidden units per layer instead of 256. Figure 4 presents the results of the different configurations.

Table 1: Parameters for training the SAC.

Parameters	Values	Parameters	Values
gamma	0.95	batch size	256
learning rate	$5 \times 10^{-4}$	buffer size	$10^5$
reward scale	2	gradient steps	16
hidden units	256	hidden layer	2
target coef.	0.005		

The task is learned for the FS problem with a mean distance of 7.29 mm for the classic SAC. However, when testing the SAC that reaches 7.29 mm on the same validation set of targets, but ending the episode when a collision occurs, the distance drops to 25.2 mm. On the second objective with a Random Start (RS), the SAC achieves an average distance of 44.1 mm which drops to 93.5 mm when adding the collision termination condition.

The alternative SAC configuration with No Collision in training ( $NC_t$ ) reaches 114.16 mm and the SAC with 16 hidden units reaches 96.14 mm. Since the performance of these alternative SAC is really bad compared to the classic SAC, they are not kept for further tests.

#### 4.3 Comparison Between the TPGs and the SAC

In this section, a comparison with the results of the SAC and the TPGs is done. A comparison of the accuracy of the algorithms is presented in Section 4.3.1 and their computational complexity are studied in Section 4.3.2.

##### 4.3.1 Performance Comparison

Table 2 shows the distance, success rate, and number of collisions achieved by both algorithms for both problems. Both the SAC with allowed collisions and

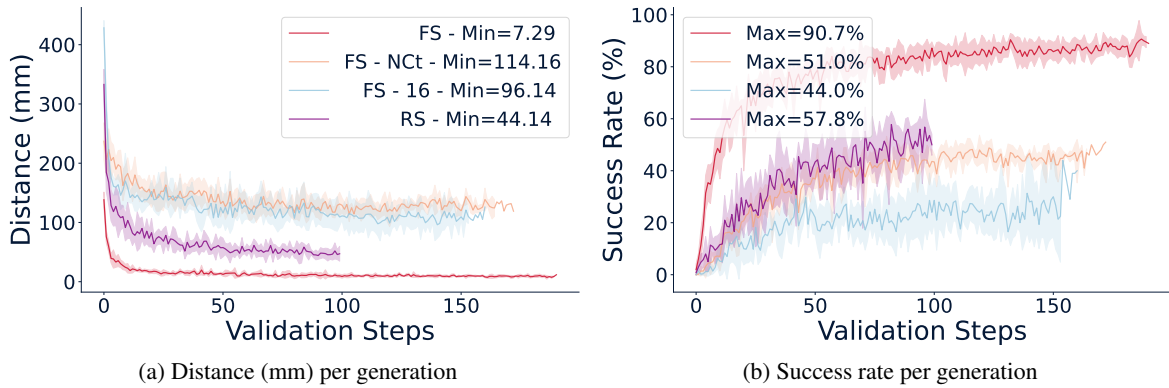


Figure 4: Results of SACs on the first objective, with fixed starting positions, and second objective, with random starting positions. The curves show the mean of the five seeds  $\pm$  the standard deviation in the shaded areas. The number 16 indicates that there are 16 number of hidden units per layer instead of 256. NC<sub>i</sub> indicates that the training is done with no collision allowed. (a) Shows the distance results. (b) Shows the percentage of success.

No Collision on inference are measured. For the distance achieved, the results are similar for the FS problem, where the SAC NC<sub>i</sub> and the TPGs reach 25.2 mm and 28.6 mm, respectively. However? The SAC NC<sub>i</sub> is still better than the TPGs for the success rate. It has a success rate of 82%, while the TPGs has 56.8%. For the RS problem, the results are better for the SAC NC<sub>i</sub> for both metrics, with a distance of 93.5 mm compared to 133.7 mm for the TPGs and a success rate of 37.4% compared to 22.2%. The only metric where the SAC and the TPGs are almost equal for both problems is the number of collisions. The SAC has 11.2% for the FS problem and 36.8% for the RS problem. The TPGs is really close with 15.5% for the FS problem and 37.2% for the RS problem.

Figures 5a and 5b show the distance reached by each algorithm and each seed for the FS and RS problems, respectively. SAC NC<sub>i</sub> has cases where the distance is really high because due to collision termination. For the FS problem, although both algorithms achieve a nearly equal mean distance, the distribution of the distance is not the same. For the SAC, most of the distances are close to 3mm, but some are really bad reaching 500mm, while for the TPGs, the distances are more spread between 3 and 200mm.

### 4.3.2 Inference Complexity Study

To measure the inference execution time, both algorithms are run on a single-CPU 13th Gen Intel(R) Core(TM) i7-13700H. SAC uses the PyTorch C++ API and the TPGs uses the C code generation proposed by GEGELATI. The code is compiled using GCC 13.3 on a Windows 10 operating system. Figures 5c and 5d show the time per action on each target for each seed for both problems. Both algorithms have similar execution times between each seed and

Table 2: Accuracy performance between the SAC over five seeds, the TPGs over 10 seeds and the hybrid solution over the 50 combined seeds. The metrics used are the distance, the success rate and the collision rate. SAC NC<sub>i</sub> refers to the SAC with no collision on inference, meaning that a collision ends the episode. The results are the mean of the seeds  $\pm$  the standard deviation.

Metrics	Distance	Success	Collision
Fixed Start Problem			
SAC	7.3( $\pm$ 03)	90.7%( $\pm$ 2)	11.2%( $\pm$ 02)
SAC NC <sub>i</sub>	25.2( $\pm$ 08)	82.0%( $\pm$ 3)	11.2%( $\pm$ 02)
TPGs	28.6( $\pm$ 09)	56.8%( $\pm$ 9)	15.5%( $\pm$ 10)
Hybrid	11.3( $\pm$ 06)	86.9%( $\pm$ 8)	12.2%( $\pm$ 08)
Random Start Problem			
SAC	44.1( $\pm$ 03)	57.8%( $\pm$ 4)	36.8%( $\pm$ 04)
SAC NC <sub>i</sub>	93.5( $\pm$ 04)	37.4%( $\pm$ 5)	36.8%( $\pm$ 04)
TPGs	133.7( $\pm$ 17)	22.2%( $\pm$ 8)	37.2%( $\pm$ 14)
Hybrid	81.2( $\pm$ 22)	49.3%( $\pm$ 9)	36.3%( $\pm$ 11)

each target for both problems. The TPGs is on average 96 times faster than SAC for both problems. The SAC takes 61.53  $\mu$ s per action, while the TPGs takes 0.64  $\mu$ s.

This complexity is also reflected in the model sizes. For the SAC, all models have the same size of 292 kB, while the models of the TPGs have a size between 11 kB and 18 kB. The models of the TPGs are 20 times smaller than the model of the SAC.

The TPGs can achieve similar performance to the SAC on this problem, while being a hundred times faster and with a model size 20 times smaller. The results of the TPGs are very promising for the trajectory planning problem. However, there are some problems with the TPGs. The TPGs, like many GP algorithms, need much more training time than Deep

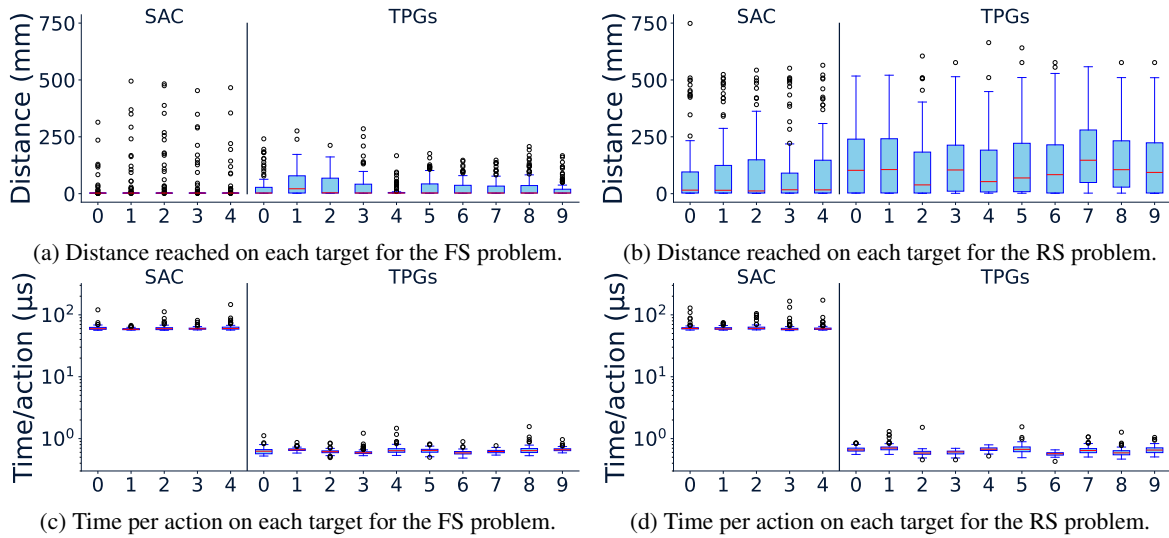


Figure 5: Results of each seed for both the SAC and the TPGs.

RL. The TPGs also have a larger standard deviation during training compared to the SAC.

## 5 HYBRID SOLUTION

In this section, we propose a hybrid version that combines the SAC and the TPG to produce a solution that surpasses the accuracy of the state-of-the-art while having a lower execution time. Section 5.1 describes the proposed algorithm, then Section 5.2 presents an experimental study to find the best solutions. Finally, Section 5.3 compares the results of the hybrid solutions with the SAC and the TPGs.

### 5.1 Hybrid Solution Description

As seen in the previous section, the TPGs and the SAC reach almost the same mean distance but the distribution of the distances is not the same. This implies that the TPGs and the SAC learn different behaviors to reach a target. The TPGs are on average 95 times less complex than the SAC. These two features motivate the hybrid solution: we want to benefit from both behaviors and from the low complexity of the TPGs.

The hybrid solution, described in Algorithm 1, relies on a TPG and a SAC that have been trained from scratch independently. For inference, the hybrid solution starts the trajectory planning with the TPG for a first approach to the target, then switches to the SAC for the finer approach. There are three conditions that cause the selection of actions to switch to the SAC:

1. If the actions of the TPG lead to a significant increase in the distance of the arm from its target.

A threshold defined by the *sepDistance* parameter is used to trigger this condition. Whenever a series of TPG actions increases the distance of the end-effector of the arm beyond this threshold, the condition is triggered and the SAC takes over the control of the robot arm.

2. If the TPG chooses the no-action option, it will terminate itself.
3. If the TPG exhibits cyclic behavior. The inference of a TPG is a deterministic and stateless process, which means that given the input state, it will always result in selecting the same action. As a consequence, a TPG can enter a cycle of actions.

If a collision occurs, no matter if the SAC or the TPG takes the action, the inference ends immediately.

After switching to the SAC algorithms, the hybrid algorithm will automatically return to the TPG control after a fixed number of actions defined by the *NbItGoBackTPG* integer parameter of the algorithm. This parameter allows to tune the ratio of actions taken by the TPG when running the hybrid algorithm.

### 5.2 Experimental Study for Hybrid Solution

The described hybrid solution depends on two parameters: *nbItGoBackTPG* the number of SAC iterations after which the TPG takes over, and *sepDistance* the distance to the target increase causing the SAC to take over. The goal of the hybrid solution is to find a compromise between accuracy and complexity. In this idea, an experimental study is carried out on the two



Algorithm 1: Inference with the TPG and the Deep RL Algorithm.

---

**Data:** *Environment*, TPG agent, SAC agent, *sepDistance*, *nbItGoBackTPG*

Initialize environment;  
 Start episode;  
 $bestDistance \leftarrow currentDistanceToTarget$ ;  
 $TPGRunning \leftarrow true$ ;  
 $counter \leftarrow 0$ ;

**while** *episode not done* **do**

**if**  $TPGRunning$  **then**

The TPG takes an action;

**if**  $bestDistance + sepDistance > currentDistanceToTarget$  **then**

$TPGRunning \leftarrow false$ ;

**else**

**if** *TPG stops moving or is cycling* **then**

$TPGRunning \leftarrow false$ ;

**end**

**end**

**else**

The SAC takes an action;

$counter \leftarrow counter + 1$ ;

**if**  $counter = nbItGoBackTPG$  **then**

$counter \leftarrow 0$ ;

$TPGRunning \leftarrow true$ ;

**end**

**end**

**if**  $bestDistance > currentDistanceToTarget$  **then**

$bestDistance \leftarrow currentDistanceToTarget$ ;

**end**

**end**

End episode;

---

parameters. 36 combinations of the parameters are tested:

- *nbItGoBackTPG*: [1, 5, 10, 50, 100, 1000];
- *sepDistance* (mm): [1, 3, 10, 20, 100,  $\infty$ ].

Infinity as the separation distance means that the SAC can only take over if the TPGs stops moving or starts cycling.

Five seeds are trained for the SAC and ten for the TPGs for each configuration. Thus, 50 pairs of the SAC and the TPG can be generated for the hybrid solution. Each of the 36 combinations is tested with the 50 combinations to ensure the viability of the results.

As shown in Section 4.3.2, the inference execution time of the TPG and the SAC have a very limited variations. Instead of the execution time, the ratio of

the actions performed by the TPG to the total number of actions performed in this episode is measured. Then, an estimated time per action is calculated with the mean time of the TPG and the SAC previously measured on the Intel i7-13700H core. This allows us to distribute this experimental study on different types of processors where the execution times may vary depending on the hardware used, while keeping a simple metric.

Figure 6 presents the results of the 36 configurations for both problems. For the FS problem, the chosen configuration is found with the *nbItGoBackTPG* set to 20 iterations and the *sepDistance* set to 100 mm. This configuration achieves an average distance of 11.3 mm with a time per action at 16.07  $\mu$ s. For the RS problem, the chosen configuration has a *sepDistance* of 100 mm and the *nbItGoBackTPG* is set at 10 iterations. This configuration reaches a mean distance of 85.5 mm with a time per action of 57.96  $\mu$ s. The two configurations are colored in green and orange, respectively, in Figure 6 to highlight that they both reached top scores within the set of configurations for both problems.

### 5.3 Comparison with State-of-the-Art

The comparison with state-of-the-art and the hybrid solutions is done in the same way as in Section 4.3, with a performance comparison in Section 5.3.1 and a complexity comparison in Section 5.3.2.

#### 5.3.1 Performance Comparison

The hybrid solution achieves better results than the SAC and the TPGs for both problems. The results are described in Table 2. For the FS problem, the mean distance of 11.3 mm is far below the mean distance of the SAC and TPG, which reached 25.2 mm and 28.6 mm, respectively. With a success rate of 86.9% the hybrid algorithm is slightly better than the SAC, with 82%. The number of collisions increased slightly from 11.2% to 12.2%, but not significantly considering the standard deviation of respectively 2% and 8% for the SAC and the hybrid solution, respectively. For the RS problem, the mean distance is better than the SAC by 13%: the SAC is at 93.5 mm while the hybrid solution is at 81.2 mm. However, the hybrid solution of this harder problem has a high standard deviation of 22 mm, while the SAC is only at 4 mm.

Figures 7a and 7b show the distance reached by the hybrid solution on different seeds compared to the SAC. For the FS problem, the hybrid solution has much fewer cases where an episode ends far from the target compared to the SAC. For the RS problem, the

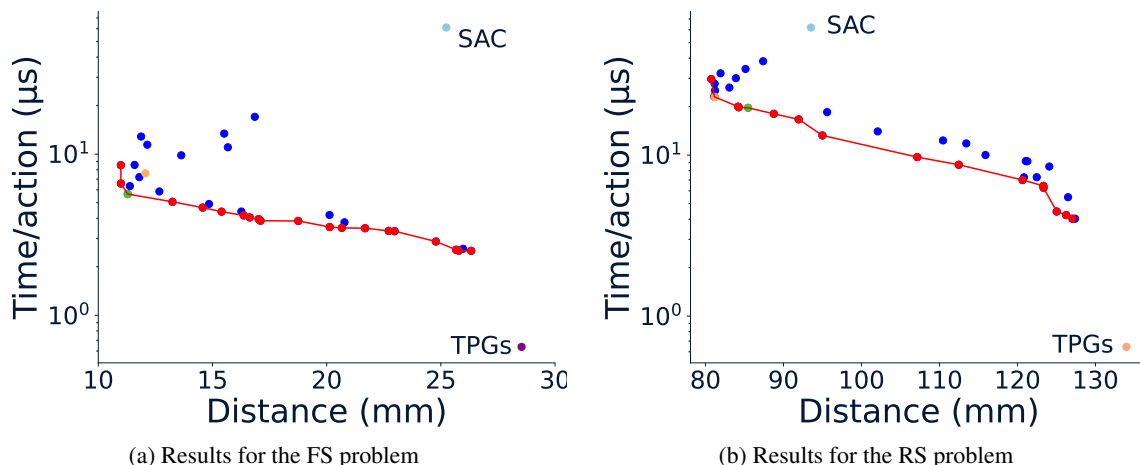


Figure 6: Pareto fronts between the distance and the approximate time per action of each of the 36 configurations of the hybrid solution for both problems. Each point is the average of the 50 combinations. The selected configuration of the FS problem is colored in green and the and the selected configuration of the RS problem is colored in orange.

distribution of the distance is better for most of the seeds for the hybrid solution than for the SAC.

To conclude this comparison, the best seeds of each solution are compared. For the FS problem, the SAC achieves a mean distance of 13.7 mm and a success rate of 87%. The hybrid solution outperforms the SAC with a success rate of 94% and a mean distance of 5.1 mm. For the RS problem, the SAC achieves a mean distance of 86.9 mm and a success rate of 33%. The hybrid solution is again much better with a success rate of 64% and a mean distance of 53.4 mm.

### 5.3.2 Complexity Study

Figures 7c and 7d show the time per action of the hybrid solutions on different SAC and TPG pairs compared to the SAC. For both problems, the hybrid solution has a better time per action with an average time of  $2.77 \mu\text{s}$  and  $10.92 \mu\text{s}$  on FS and RS problems respectively while the SAC has a time per action of  $61.5 \mu\text{s}$  for both problems. The hybrid solution is more effective, by being twenty times faster than the SAC for the FS problem and 6 times quicker of the RS problem. The best seeds of the hybrid solution detailed at the end of the previous section have a time per action of  $3.05 \mu\text{s}$  for the FS problem and  $11.55 \mu\text{s}$  for the RS problem. Because the hybrid solution is a combination of the TPGs and the SAC, the hybrid solution is slower than the TPGs. The TPGs are 4 times faster than the hybrid solution on the FS problem and 17 times faster on the RS problem.

For the SAC, all models have the same size of 292 kB, while the models of the TPGs have a size between 11 kB and 18 kB. Thus, the hybrid solution has a model size of about 305 kB.

## 6 DISCUSSION

Beyond the performance of the hybrid solution, the contribution of this work lies in the significant advantages obtained by combining the TPG and the deep RL algorithm. We show that the synergistic effects of their complementary strengths lead to improved results in robot control, in accuracy, and especially in complexity.

This combined approach opens up new possibilities for solving robot control problems. With the evolution of deep RL algorithms, harder robot control tasks are being learned every day (Harnoja et al., 2024), with larger model sizes, leading to complex solutions. Recent research explores reduction in computational cost of the deep RL agent (Gu et al., 2023) and is complementary to our approach. A hybrid solution using a TPG combined with a low complexity deep RL algorithm could give even better results.

The task studied in this paper, trajectory planning, is well suited for the hybrid solution because it is easy to detect when a TPG has bad behavior. Better ways to detect these bad behaviors should be investigated in future work to adapt this hybrid solution to a wider variety of RL tasks where the TPG have recently been applied (Kelly et al., 2021; Smith and Heywood, 2023). The TPG have also been applied to classification tasks (Smith et al., 2021; Chillet et al., 2023). Studying the design of a hybrid solution for classification tasks using the TPG and some deep learning techniques could also lead to interesting results.

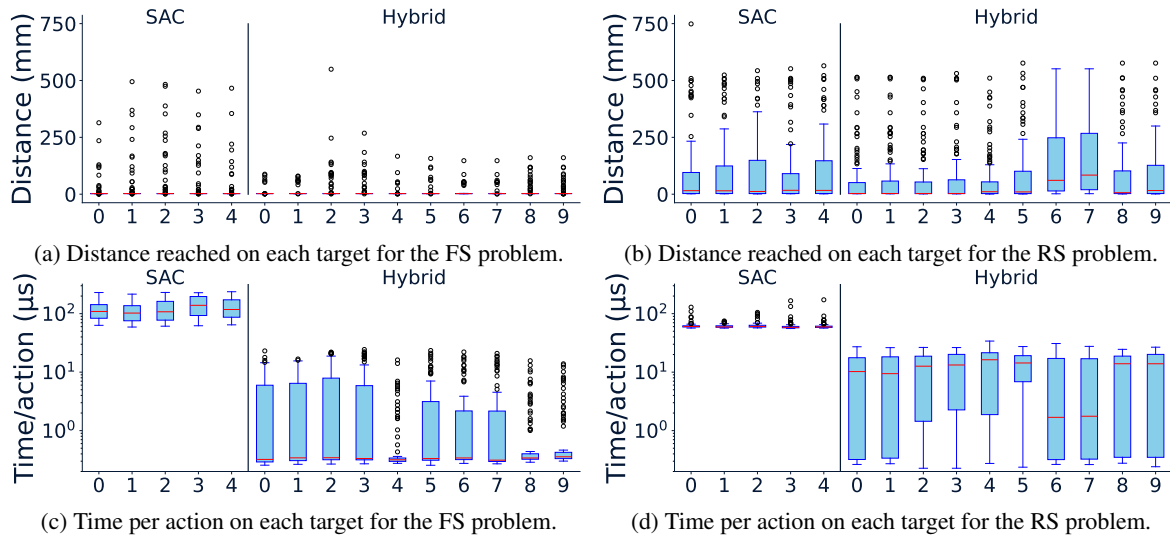


Figure 7: Results of 10 combined seeds of the best hybrid solution for both problems. Plotting the 50 combined seeds would have been unreadable, instead 10 combinations are shown, the first two seeds of the TPGs with the first of the SAC, seeds 2 and 3 of the TPGs with the second of the SAC, and so on.

## 7 CONCLUSION

This paper presents a low complexity solution for trajectory planning on IK using a genetic RL algorithm. The adaptability of the TPGs allows the learning of the complex environment. An empirical study of the parameters of the TPG makes it possible to achieve almost state-of-the-art performance on a problem, while being 100 times faster and having a model size 20 times smaller. However, the tuning is not sufficient to learn a harder problem, where even the state-of-the-art struggles. In order to propose a solution with low complexity and to increase the accuracy on this problem, a hybrid solution combining the SAC, a state-of-the-art algorithm, and the TPGs is designed. It achieves a better accuracy than both SAC and the TPGs for both problems, while being 20 to 6 times faster than the SAC depending on the problem. This hybrid solution does not use any additional training, the solution uses pre-trained models of the SAC and the TPGs. Designing training for the hybrid solution could improve the results and reduce the running time.

For other IK applications, RL algorithms use memory, continuous action, multi-action, and visual tracking, four capabilities that the implementation of the TPGs used in this paper does not have. An extension of the TPGs using memory and continuous action has already been proposed (Kelly and Banzhaf, 2020). Future work will consider these extensions, as well as multiple actions, to improve performance for the IK problem.

## ACKNOWLEDGEMENTS

This research was funded, in whole or in part, by the Agence Nationale de la Recherche (ANR), grant ANR-22-CE25-0005-01. A CC BY license is applied to the AAM resulting from this submission, in accordance with the open access conditions of the grant.

## REFERENCES

- Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.
- Chillet, A., Boyer, B., Gerzaguet, R., Desnos, K., and Gautier, M. (2023). Tangled program graph for radio-frequency fingerprint identification. In *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–7. IEEE.
- Desnos, K., Sourbier, N., Raumer, P.-Y., Gesny, O., and Pelcat, M. (2021). Gegalati: Lightweight artificial intelligence through generic and evolvable tangled program graphs. In *Workshop on Design and Architectures for Signal and Image Processing (14th edition)*, pages 35–43.
- Duka, A.-V. (2014). Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. *Procedia Technology*, 12:20–27. The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania.
- Giorelli, M., Renda, F., Calisti, M., Arienti, A., Ferri, G., and Laschi, C. (2015). Neural network and jacobian method for solving the inverse statics of a cable-driven

- soft arm with nonconstant curvature. *IEEE Transactions on Robotics*, 31(4):823–834.
- Gu, S., Kuba, J. G., Chen, Y., Du, Y., Yang, L., Knoll, A., and Yang, Y. (2023). Safe multi-agent reinforcement learning for multi-robot control. *Artificial Intelligence*, 319:103905.
- Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. (2019). Learning to walk via deep reinforcement learning.
- Haarnoja, T., Moran, B., Lever, G., Huang, S. H., Tirumala, D., Humplik, J., Wulfmeier, M., Tunyasuvunakool, S., Siegel, N. Y., Hafner, R., et al. (2024). Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 9(89):eadi8022.
- Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.
- Ho, C.-K. and King, C.-T. (2022). Automating the learning of inverse kinematics for robotic arms with redundant dofs.
- Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. (2021). How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4–5):698–721.
- Kelly, S. (2018). *Scaling genetic programming to challenging reinforcement tasks through emergent modularity*. PhD thesis, Dalhousie University, Halifax, Nova Scotia, Canada.
- Kelly, S. and Banzhaf, W. (2020). Temporal memory sharing in visual reinforcement learning. *Genetic Programming Theory and Practice XVII*, pages 101–119.
- Kelly, S. and Heywood, M. I. (2017). Emergent tangled graph representations for atari game playing agents. In *Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings 20*, pages 64–79. Springer.
- Kelly, S., Voegerl, T., Banzhaf, W., and Gondro, C. (2021). Evolving hierarchical memory-prediction machines in multi-task reinforcement learning. *Genetic Programming and Evolvable Machines*, 22:573–605.
- Phaniteja, S., Dewangan, P., Guhan, P., Sarkar, A., and Krishna, K. M. (2017). A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots. In *2017 IEEE international conference on robotics and biomimetics (ROBIO)*, pages 1818–1823. IEEE.
- Sciavicco, L. and Siciliano, B. (2012). *Modelling and control of robot manipulators*. Springer Science & Business Media.
- Smith, R. J., Amaral, R., and Heywood, M. I. (2021). Evolving simple solutions to the cifar-10 benchmark using tangled program graphs. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 2061–2068. IEEE.
- Smith, R. J. and Heywood, M. I. (2023). Interpreting tangled program graphs under partially observable dota 2 invoker tasks. *IEEE Transactions on Artificial Intelligence*, 5(4):1511–1524.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Vacher, Q., Beuve, N., Allaire, P., Marty, T., Dardaillon, M., and Desnos, K. (2024). Ecta24 artifacts.
- Wampler, C. (1986). Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *Systems, Man and Cybernetics, IEEE Transactions on*, 16:93 – 101.
- Wolovich, W. A. and Elliott, H. (1984). A computational technique for inverse kinematics. *The 23rd IEEE Conference on Decision and Control*, pages 1359–1363.
- Zhong, J., Wang, T., and Cheng, L. (2021). Collision-free path planning for welding manipulator via hybrid algorithm of deep reinforcement learning and inverse kinematics. *Complex & Intelligent Systems*, pages 1–14.