USE OF EVOLUTIONARY ALGORITHMS FOR STATIC TRAJECTORY PLANNING

Marek Paták

Faculty of Mechanical Engineering, Brno University of Technology
Institute of Automation and Computer Science
Technicka 2896/2, Brno 616 69, Czech Republic
217872@vutbr.cz

Abstract: *This paper investigates the optimization of robotics arms in static trajectory tasks using Evolutionary Algorithms to address the challenges of Inverse Kinematics problematics. It focuses on various Evolutionary algorithms methods like Genetic Algorithms, Particle Swarm Optimization, Gravitational Search Algorithms, and Differential Evolution. Describing how they can be used to solve Inverse Kinematics with different optimization parameters*

Keywords: *Evolutionary algorithms, Genetic Algorithms, Particle Swarm Algorithm, Gravitational Search Algorithm, Differential Evolution, Inverse Kinematics, Forward kinematics*

1 Introduction

Recently, six-axis robots have become a common part of most production halls. Despite marketing demos of various interesting use cases, pick-place applications along an unchanged trajectory are still the most common use cases that are being in production. And even for these basic applications we still haven't found an optimal solution [3]. The goal for this application would be to follow a given trajectory precisely with movement that will do the least harm to the robot and also will cost the least electricity. In recent years Evolution Algorithms seemed to have promising results in this kind of multi-criteria optimization [11].

2 Inverse Kinematics (IK)

IK is a critical process in robotics essential for controlling the movement of robotic arms. While forward kinematics involves calculating the position of end effectors based on known joint angles, IK solves the opposite problem. It finds the necessary joint angles that will achieve the desired position of the end effector. This reversal from forward to IK introduces several complex challenges. One is that the complexity of this problem grows exponentially with the amount of degrees of freedom that need to be controlled. Because for each point in the trajectory, several possible configurations can achieve this point. Another is that not every point is reachable depending on the geometry of the robot you might not be able to reach the point because of distance or collisions. The computed configurations have to also be checked for the singularities of the robot. To address these challenges, various methods have been developed. One common approach is the use of numerical methods, like the Jacobian Pseudo-inverse and Jacobian Transpose techniques. These methods iteratively adjust joint angles to minimize the difference between the current and desired end effector positions [1].

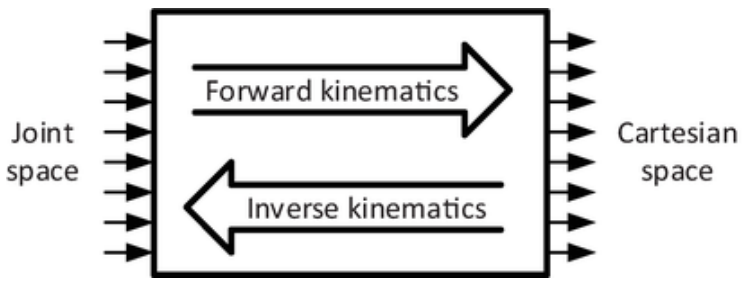

Figure 1: Difference between Joint and Cartesian space [15]

3 Evolutionary Algorithms (EA)

EA represents a class of optimization techniques inspired by the process of natural selection. These algorithms are particularly adept at solving complex, non-linear problems like those found in IK. EA operates on a population of potential solutions, iteratively improving them through processes analogous to genetic mutation, crossover, and selection. In the context of IK, these algorithms offer a robust way to explore the solution space, often leading to effective solutions where traditional methods struggle [20].

3.1 Genetic Algorithms (GA)

GA, first emerging in the 1990s as one of the initial forms of evolutionary algorithms [9, 14], have since gained prominence due to their effectiveness and versatility. These algorithms work based on human chromosomes, where they start with a population of randomly generated solutions. As this population evolves over successive generations, each solution is scrutinized under a fitness function. In the realm of inverse kinematics, this function is crucial as it quantifies how precisely a joint configuration matches the desired end effector position. Solutions that achieve closer alignment with the target receive higher fitness scores, denoting their superiority in solving the IK challenge. The most fit among these are selected preferentially for the breeding process, during which genetic operators such as crossover and mutation are applied to produce new offspring. This selective breeding and genetic modification echo the principles of natural selection, progressively guiding the population towards more optimal solutions. A key aspect of GA is its approach to handling suboptimal solutions. Rather than discarding these solutions outright, they are retained within the algorithm's population. This retention is crucial as it contributes to the genetic diversity of the population. Maintaining a diverse set of solutions is essential in preventing the algorithm from prematurely converging on local minima. These local minima are solutions that may seem optimal within a limited perspective but are inferior when considering the entire solution space [17].

3.2 Particle Swarm Optimization (PSO)

PSO is a computational method inspired by the social behavior of birds flocking or fish schooling. It's particularly effective for solving optimization problems in continuous spaces. PSO operates by initializing a group of candidate solutions, known as particles, which explore the solution space by following the current optimal solutions. Each particle in PSO has two essential attributes: its position, which represents a potential solution, and its velocity, which determines how fast and in what direction the particle moves through the solution space. The algorithm updates these attributes iteratively based on two criteriums: the best position encountered by the particle itself and the best position encountered by the entire swarm.

The update rules for a particle's velocity and position can be represented by the following equations:

Velocity update:

$$v_i^{(t+1)} = w \cdot v_i^{(t)} + c_1 \cdot r_1 \cdot (p_{best,i} - x_i^{(t)}) + c_2 \cdot r_2 \cdot (g_{best} - x_i^{(t)}), \tag{1}$$

Where:

$v_i^{(t+1)}$ is the new velocity of particle i at time $t+1$,
w is the inertia weight that controls the impact of the previous velocity,
c_1 and c_2 are cognitive and social coefficients,
r_1 and r_2 are random numbers between 0 and 1,
$p_{\text{best},i}$ is the personal best position of particle, i.
g_{best} is the global best position found by the swarm,
$x_i^{(t)}$ is the current position of particle i at time t.

Position update:

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}, \tag{2}$$

Where:

$x_i^{(t+1)}$ is the new position of particle i at time $t+1$.

In PSO, each particle adjusts its trajectory towards its personal best and the global best positions. The

inertia weight w balances the exploration and exploitation. A high w facilitates exploration areas, while a low w encourages exploitation, focusing on current promising areas. The cognitive coefficient c_1 represents the particle's own experience, while the social coefficient c_2 represents the collaborative aspect of the swarm. The strength of PSO lies in its simplicity and its ability to efficiently converge to and optimal or near-optimal solution. It's widely used for various optimization problems in engineering and science due to its efficacy in handling complex, multidimensional spaces [13, 10].

3.3 Gravitational Search Algorithm (GSA)

GSA, as developed by Rashedi et al.[6], is an innovative optimization technique inspired by the laws of Newtonian physics, specifically gravity and motion. The fundamental idea is to mimic the way objects interact in a gravitational field within the context of a search problem. In GSA, each potential solution to the optimization problem is represented as a particle or mass in a multidimensional search space. The position of each mass signifies a possible solution, and the fitness function of the solution is akin to the mass's weight: the better the solution, the heavier the mass. The algorithm employs the concepts of Newton's law of gravitation and motion. According to these laws, every mass exerts a gravitational force on every other mass. In GSA, this translates to each potential solution influencing others based on their fitness. Better solutions have a stronger gravitational pull, attracting other solutions towards them. As the algorithm progresses, masses move towards the positions of heavier masses. This movement is guided by the forces of attraction and the mass's inertia representing the resistance to change. The gravitational constant, a crucial part of Newton's law, is also a key parameter in GSA. It starts with a large value to allow global exploration of the search space and decreases over time, enabling a gradual shift to a more intensive local search around the current best solutions [2]. GSA seems to get slower to convergence but may have some potential in solving specific problems [8].

3.4 Differential Evolution (DE)

DE is an evolutionary algorithm that simulates the process of natural selection. Its primary goal is to incrementally refine a set of potential solutions. During each iteration of the algorithm, DE creates new potential solutions [19]. It does this by blending elements from three distinct and randomly selected individuals from the existing population. The algorithm then assesses these new solutions against the current ones using a predefined fitness criterion. If a new solution demonstrates better fitness, it replaces its predecessor in the population. What makes DE particularly versatile and effective is its use of mutation and crossover techniques. These techniques allow the algorithm to thoroughly explore and navigate the solution space, searching for the most optimal solutions to complex problems [4].

The whole process can be divided into the following parts:

Population Initialization:

$$X_i^{(0)} = x_i^{(0)} \tag{3}$$

Mutation (Differential Variant):

$$V_i^{(g+1)} = X_{r1}^{(g)} + F \cdot (X_{r2}^{(g)} - X_{r3}^{(g)}) \tag{4}$$

Crossover (Exponential Variant):

$$U_{i,j}^{(g+1)} = \begin{cases} V_{i,j}^{(g+1)} & \text{if } rand_j \leq CR \text{ or } j = j_{rand}, \\ X_{i,j}^{(g)} & \text{otherwise,} \end{cases} \tag{5}$$

Selection (Elitism):

$$X_i^{(g+1)} = \begin{cases} U_i^{(g+1)} & \text{if } f(U_i^{(g+1)}) < f(X_i^{(g)}), \\ X_i^{(g)} & \text{otherwise.} \end{cases} \tag{6}$$

Where:

$f(\cdot)$ is the fitness function,
i is an index, with $i = 1, 2, \ldots, N$,
$r1, r2,$ and $r3$ are randomly selected indices from the population,
F is the mutation factor,
$rand_j$ represents random values for each element j of the vector,
CR is the crossover constant,
j_{rand} is a randomly selected index [11].

4 Use Cases

Evolutionary algorithms are particularly effective for solving static trajectories where computation time is not a primary concern due to their ability to explore a vast range of possible solutions and adaptively refine them. This approach is ideal for static trajectory optimization because it allows for a thorough examination of various paths, leading to an optimal or near-optimal solution that might not be easily discovered through conventional methods. Moreover, the flexibility of evolutionary algorithms makes them well-suited for adapting to different constraints and requirements, ensuring a highly tailored solution for specific trajectory problems.

4.1 Reduction in Energy Consumption

EA are essential tools for enhancing the energy efficiency of six-axis robots in production settings, leading to cost reductions and sustainability improvements [12]. By optimizing movement trajectories and fine-tuning speed and acceleration profiles, EA minimizes unnecessary energy consumption while ensuring efficient operation. This not only lowers operational costs but also extends the lifespan of robots. In general, EA cloud plays a crucial role in driving down energy efficiency, cost-effectiveness, and environmental responsibility in manufacturing processes [18, 7].

4.2 Reduction in Cycle Time

EA can also be used in problems where the goal of the optimization is to reduce cycle time. This concept has been part of the debate for several years [16] and still holds its attention [5].

4.3 Special

There are also special task that EA are proving to work great such as welding [20] and painting [21].

Figure 2: Welding process generated by EA [20]

5 Conclusion

This paper explored the application of EA for optimizing the performance of industrial robots for static trajectories, which involves solving the IK problem which was outlined in 2. In the next chapter, we took a detailed look at how GA, PSO, GSA, and EA can address this issue. And in the last section 4 some of the real-world use cases were noted

References

[1] Aristidou, A., and Lasenby, J. Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver. *Research gate* (2009).

[2] Ayyildiz, M., and Çetinkaya, K. omparison of four different heuristic optimization algorithms for the inverse kinematics solution of a real 4-dof serial robot manipulator. *Neural Comput Applic* (2016).

[3] Billard, A., and Kragic, D. Trends and challenges in robot manipulation. *Science* (2019).

[4] C. Gonzalez, D. B., and Moreno, L. Optimum robot manipulator path generation using differential evolution. *IEEE Congress on Evolutionary Computation* (2009).

[5] E. Ferrentino, A. Della Cioppa, A. M., and Chiacchio, P. An evolutionary approach to time-optimal control of robotic manipulators. *J Intell Robot Syst* (2020).

[6] E. Rashedi, H. N.-p., and Saryazdi, S. Gsa: A gravitational search algorithm. *Information Sciences* (2009).

[7] et al., A. V. Reduction in robotic arm energy consumption by particle swarm optimization. *Applied Sciences* (2020).

[8] G. Kanagaraj, S. S. M., and Yu, V. F. Meta-heuristics based inverse kinematics of robot manipulator's path tracking capability under joint limits. *MENDEL* (2022).

[9] J. K. Parker, A. R. K., and Goldberg, D. E. Inverse kinematics of redundant robots using genetic algorithms. *IEEE Computer Society* (1989).

[10] L. Yiyang, J. Xi, B. H. W. Z., and Liangliang, S. A general robot inverse kinematics solution method based on improved pso algorithm. *IEEE Access* (2021).

[11] M. Juricek, R. P., and Kudela, J. Evolutionary computation techniques for path planning problems in industrial robotics: A state-of-the-art review. *Computation* (2023).

[12] M. Soori, B. A., and Dastres, R. Optimization of energy consumption in industrial robots, a review. *Cognitive Robotics* (2023).

[13] N. Rokbani, B. Neji, M. S. S. M., and Ghandour, R. A multi-objective modified pso for inverse kinematics of a 5-dof robotic arm. *Applied Sciences* (Applied Sciences).

[14] Nearchou, A. C. Solving the inverse kinematics problem of redundant robots operating in complex environments via a modified genetic algorithm. *Mechanism and Machine Theory* (1998).

[15] O. Hock, J. Šedo, O. H., and Šedo, J. *Forward and Inverse Kinematics Using Pseudoinverse and Transposition Method for Robotic Arm DOBOT*. IntechOpen, 2017.

[16] Rana, A. S., and Zalzala, A. M. S. Near time-optimal collision-free motion planning of robotic manipulators using an evolutionary algorithm. *Robotica* (1996).

[17] S. Momani, Z. A.-H., and Alsmadi, P.-O. Solution of inverse kinematics problem using genetic algorithms. *Applied Mathematics Information Sciences* (2015).

[18] S. Števo, I. S., and Dekan, M. Optimization of robotic arm trajectory using genetic algorithm. *IFAC Proceedings Volumes* (2014).

[19] Storn, R., and Price, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* (1997).

[20] X. Liu, C. Qiu, Q. Z. A. L., and Xie, N. Time-energy optimal trajectory planning for collaborative welding robot with multiple manipulators. *Procedia Manufacturing* (2020).

[21] X. Liu, Y. Feng, Q. Z. X. H., and Yang, Z. Multi-objective optimization of spraying trajectory planning for large ship blocks using evolutionary computation. *Procedia CIRP* (2021).