Robotic Arm Kinematics : Point Swarm Optimization

Steven VanCamp
Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824

(Dated: May 1, 2023)

I. ABSTRACT

In the field of robotics, solving the inverse kinematics problem is a complicated but necessary task. In many cases it has no analytical solution and must be computed numerically. This requires computation resources and time. When time is a constraint there are various techniques to decrease the computation time, such as parallel computing. Parallel computing utilizes modern CPU architecture to run multiple programs at once, which may communicate with each other. By combining an inverse kinematics solver, like a point swarm optimization algorithm with parallel computing its possible to achieve some degree of speedup. Some CPU configurations may not provide any speedup and would benefit by running in serial, but this is not always the case. Results show that for some cases a 33% speedup in the time it takes to converge to an acceptable solution is possible.

FIG. 1. The robotic arm that has been the focus of this project

II. INTRODUCTION

The integration of robotics into our everyday life requires that we can quickly and accurately direct robots of varying designs to specific positions and configurations. Without the ability to do so could result in injury or damage to people or property. As such having robust control schemes and algorithms that can provide accurate solutions for these directives quickly is critical. In addition, these algorithms are also necessary for various robotics applications in a setting where the environment around a robotic element may be changing, such as in a factory or manufacturing facility.

The focus of this project was on exploring one such algorithm and whether parallelization of the algorithm

would provide any speedup, while maintaining the same degree of accuracy in the final solution. The algorithm in question is a Point Swarm-Optimization (PSO) based algorithm for Inverse Kinematics (IK) of a robotic arm as discussed in [1]. IK is a general approach to finding intermediate positions or angles of robotic joints between the "base" and the end-effector. The main problem with IK, is that it generally does not have an analytical solution. PSO is an optimization algorithm that populates the parameter space with particles representing the parameters and finds a solution to a given problem.

As discussed in [1] PSO can be effectively applied to the IK problem of robotic arms to find the intermediate joint angles for a desired end-effector position. However, [1] does not go into the timing of such an algorithm and purely evaluates the evolution of the algorithm based on the number of iterations it takes to find a solution below some accuracy threshold. This is what I plan to explore, the timing of the algorithm and whether a parallel form of the algorithm could provide a speedup.

III. METHODOLOGY

The overall focus of this project was on attempting to speedup the PSO algorithm used to solve the IK problem of a robotic arm as discussed in [1]. To do this various components of the algorithm had to be written and validated prior to attempting to write the parallel version of the algorithm. The major components f the algorithm are discussed below (III A, III B). These were written for this project, as finding and rewriting an open-source was decided to be to time consuming. This was in-part because the algorithm discussed in [1] is fairly new (2019).

A. Forward Kinematics

As part of the algorithm discussed in [1] the algorithm must be able to solve the Forward Kinematics of the robotic arm. FK is a problem where the intermediate joint positions & angles are known and the end-effector position must be calculated. In contrast to IK, FK generally has an analytical solution and can be solved quite efficiently. The FK method used in this project is based on the Denavit Hartenberg Representation (D-H) [2] of the robotic arm in question. With this representation and a bit of linear algebra the end-effector position can be solved for any value of the intermediate joint position or angle.

In the D-H representation the robotic arm is represented as a series of links and joints, each with a corresponding matrix that describes its physical dimensions and orientation A_i. Generally each A_i matrix has 1 parameter that is allowed to vary q_i, be it a rotation or a linear motion. To find the position and orientation of the end-effector Eq.1 is used to find the translation matrix corresponding to the end-effector and from it the position and orientation can be extracted.

$$T_n = A_0(q_0)A_1(q_1)...A_{n-1}(q_{n-1})A_n(q_n) \tag{1}$$

B. Point Swarm Optimization

Point swarm optimization is method of optimization that is a derivative free and is population based search optimization method. PSO populates the search space with particles, each representing 1 solution to the problem at hand. They move throughout the parameter space with a velocity vector defined by a random component and is adjusted by inter-particle relationships and historical information [3]. The exact method by which each particles velocity is adjusted depends on the problem itself. In many cases this adjustment is based on a fitness evaluation of each particle's solution.

The PSO algorithm used in this project is discussed in [1], and shown in Fig.6. In this project each particle had an associated set of values representing the joint angles being optimized $\theta_{1..6}$. The fitness of each particle was evaluated by calculating the end-effector position & orientation given the particles parameter values $\theta_{1..6}$. The end-effector position & orientation was then compared to the desired state of the end-effector and a fitness value was assigned, a lower fitness value represents a better solution.

C. Parallelization: OpenMP

For this project OpenMP was chosen as the parallel implementation due to its shared-memory framework. The PSO algorithm used in this project could be written with either a shared-memory or distributed-memory framework. However, it is necessary for threads to be able compare the current best solution and exchange

information on this. As such a shared-memory framework would increase the parallel efficiency by reducing the communication time, and taking advantage of the memory memory structure of the hardware. There is also the fact that in practice a robot will likely have its own onboard processing unit and memory, a distributed-memory framework does not make sense in this case.

The algorithm used in this project, shown in Fig.6, requires little modification to run in parallel using the OpenMP implementation. The loop defined by the upward arrow on the right is the main loop to be parallelized. Careful considerations must be made about what elements are private or shared, and which must be vectorized in order to maintain a thread-safe environment. The basic structure of the parallel program is the following:

1. Each thread is assigned a portion of the particle population to handle
2. Each thread maintains a personal best particle for each iteration
3. After each iteration the personal best particle is compared between each thread
4. The global best particle is chosen to be the previous global best particle or one of the personal best particles if it has a better fitness
5. The next iteration is run

IV. RESULTS

Once the necessary portions of the code had been written they were integrated and tested. This was done to ensure no race conditions were meet and the algorithm behaved normally. The program was tested using the same setup and values as described in [1], section IV. A. This was done so that the results of the program could be continuously monitored to ensure the program was working. In addition to this, this case was used since the solution is not trivial and a comparison could be made between my program and the results presented in [1]. Of note all data was generated when running the program on my personal laptop, the hardware specifications are shown in B.

The performance of the serial algorithm is shown in Fig.2. This is a very similar result to Fig.6 in [1]. Both figures were made with data from runs that were solving the same IK problem. Comparing the two figures shows that the results from my program are qualitatively similar to the results attained in the original paper. This is important as it shows the algorithm is *likely* working as intended. Without a quantitative comparison to the original data or algorithm its uncertain whether they are performing in exactly the same manner. However, for the purpose of this project a qualitative comparison should be sufficient. Having shown the program is operating as expected, the rest of this section will focus on the parallel performance.

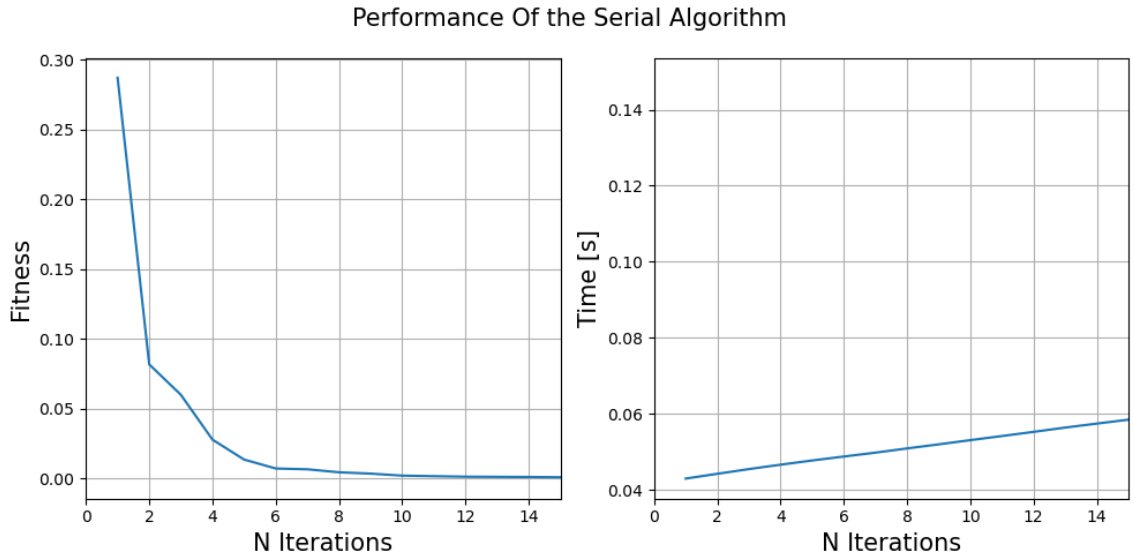

FIG. 2. Serial performance of the algorithm. For the right graph the y-axis represents the mean time since the start of the program. Each point represents the mean value over 100 runs. Each run was made with 2000 points in the point swarm.

We first look at how increasing the number of threads affects the performance of the algorithm. Fig.3 shows the mean fitness of the global best particle across all threads at each iteration, the mean taken over 100 sample runs. The results shown here seem to indicate that there is an optimal number of threads to run the program with, if the fitness is the only concern. The 8-thread program starts at a lower fitness value, and seems to maintain this "lead" as the program iterates.

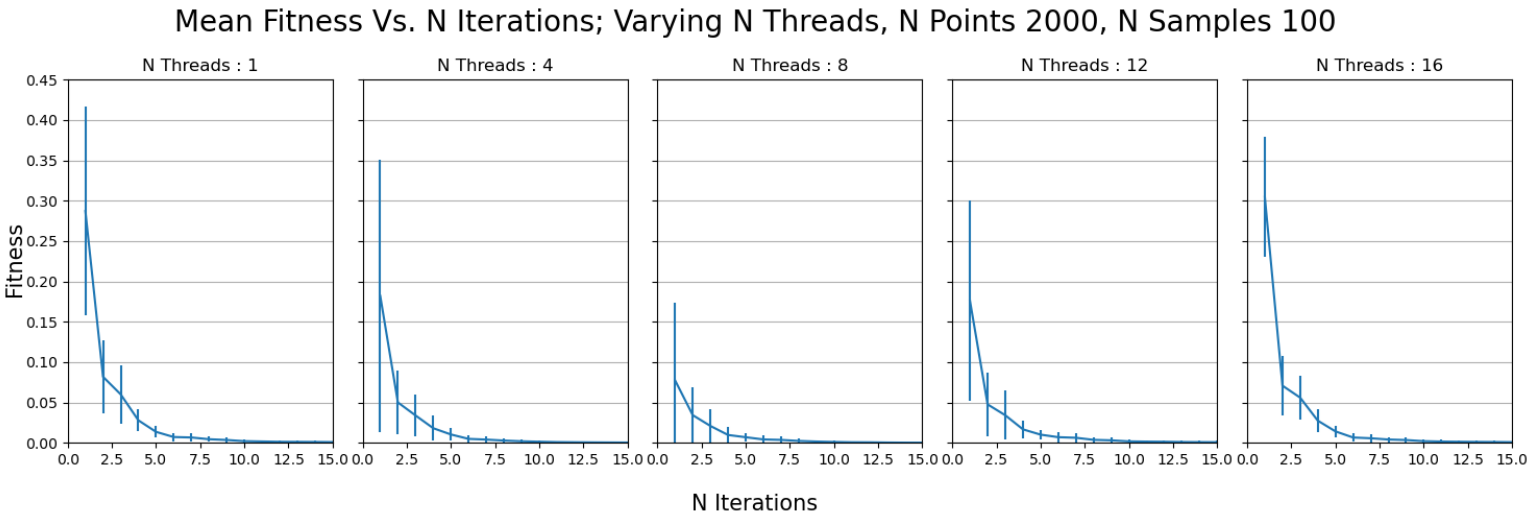

FIG. 3. Global Best Particle Mean Fitness Vs. Iteration Number. Each point represents the mean value over 100 runs. Each run was made with 2000 points in the point swarm.

For a situation where there is no time constraint this is an important result. If the only concern is the most accurate solution then this could inform the user how many threads to dedicate to this process. However, its important to remember that the purpose of this project is to find an optimal solution to the robotic arm IK problem in as short a time as possible. As such the time it takes to converge to a solution that meets some accuracy criteria is arguably more important.

Fig.4 shows the time it takes for the program to converge to a solution below a fitness value of $1E-6$. This shows that running the program in parallel may result in a faster time to converge to a solution below some accuracy. In this case the program consistently converges faster when running with $[4, 14]$ threads. The time to converge does vary more in this region, as shown by the error bars which represent the 1σ range of convergence times.

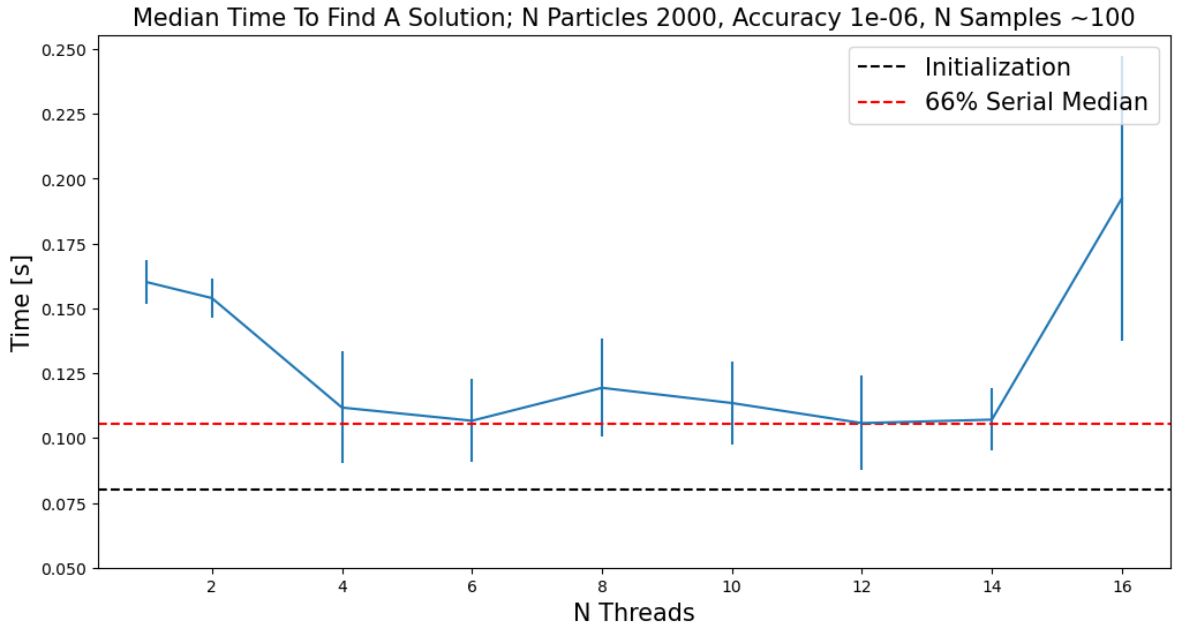

FIG. 4. Convergence Time VS. N Threads. Each point represents the median value over 100 runs. The dashed black line shows the mean time it takes for the program to initialize.

Fig.5 shows how the algorithm behaves when the number of points in the point swarm increases. Rather than breaking once a solution below a given accuracy was achieved, the program was allowed to run for the maximum number of iterations. Qualitatively the program's execution time approximately doubles as the number of particles doubles. This result is expected, but important to verify. As the number of particles increases its expected the time to converge to a solution below some accuracy decreases. However, no testing was done to confirm this.

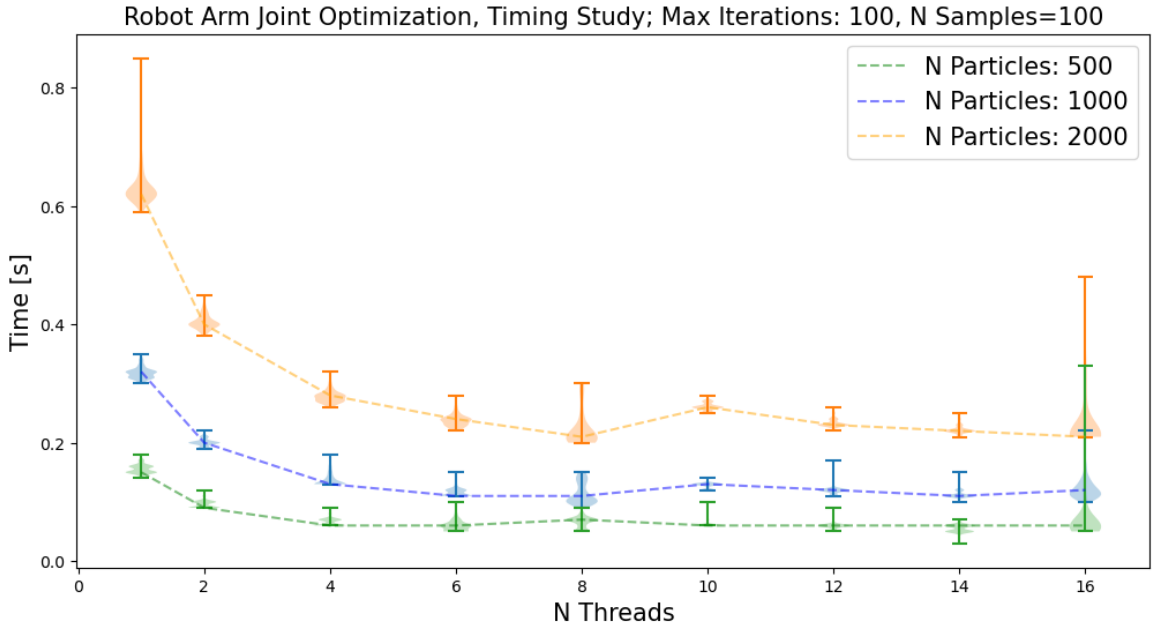

FIG. 5. Time to complete 100 iterations VS. N Threads. The dashed lines pass through the median values of the distributions shown by the violin plots. Each violin plot shows the distribution of the execution times.

V. CONCLUSION

By utilizing a parallel version of a PSO algorithm to solve an IK problem it has been shown that there is a potential 33% speedup in convergence time over the serial version of the same PSO IK program, 4. This is not always true, in some cases the convergence time is slower than when the program is run in serial; 4, 3. Since physical systems may change and computational resources are varied in the field of robotics, testing different configurations is the best method to determine the optimal setup for a given scenario.

VI. REFERENCES

[1] M. Alkayyali and T. A. Tutunji, in *2019 20th International Conference on Research and Education in Mechatronics (REM)* (2019) pp. 1–6.
[2] M. V. Mark W. Spong, Seth Hutchinson, *Robot Modeling and Control, 2nd Edition* (Wiley, New York, 2020).
[3] B.-I. Koh, A. D. George, R. T. Haftka, and B. J. Fregly, International Journal for Numerical Methods in Engineering **67**, 578 (2006), https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1646.

Appendix A: Images

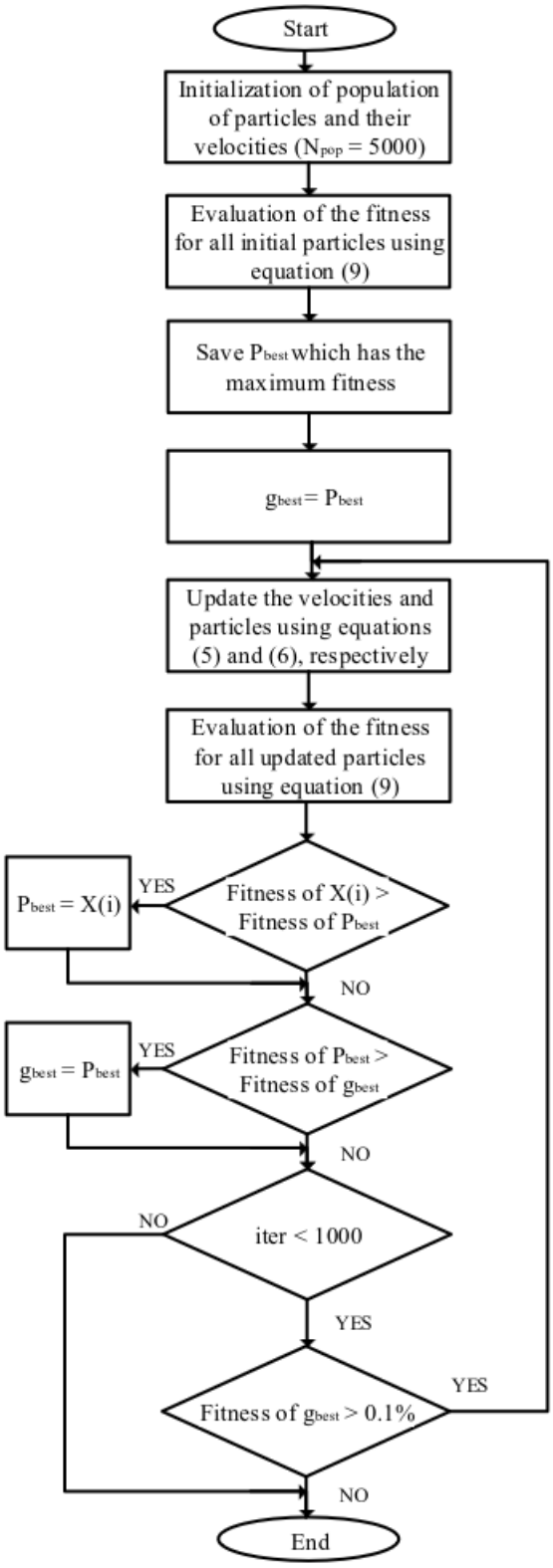

FIG. 6. General approach to the PSO based algorithm used to solve the IK problem for a robotic arm. [1]

Appendix B: Hardware

Personal Laptop:
CPU:

- CPU(s): 16
- Vendor ID: AuthenticAMD
- Model name: AMD Ryzen 7 5825U with Radeon Graphics
- Thread(s) per core: 2
- Core(s) per socket: 8
- Socket(s): 1
- Frequency boost: enabled
- CPU max MHz: 4546.8750
- CPU min MHz: 1600.0000
- BogoMIPS: 3992.49

Memory:

- L1: 256 KiB (8 instances) - clock (1GHz)
- L2: 4 MiB (8 instances) - clock (1 GHz)
- L3: 16 MiB (1 instance) - clock (1 GHz)
- RAM: 8 GiB (2 instances) - clock (3200MHz)