Parallel Computing - Midterm Report

Inverse Kinematics Solver

Student: Hsin-Yu Chen（陳昕佑）
ID: 61375017H

Introduction

Inverse Kinematics (IK) is essential in robotics, enabling the calculation of joint parameters that achieve desired end-effector positions and orientations. As computational demands grow, especially for real-time control or large-scale simulations, leveraging multicore processors becomes increasingly important. Brute-force IK, though conceptually straightforward, can become prohibitively expensive due to the sheer number of calculations. Hence, the goal is to evaluate whether parallelization significantly reduces computation time. I implement and benchmark single-threaded, Pthreads, and OpenMP versions of a C-based 6-DOF arm IK solver, comparing performance when processing thousands of waypoints. Advanced collision checks and real-time testing remain outside this project's scope.

Related Work & Literature Review

Inverse Kinematics (IK) methods range from closed-form, analytic solutions that directly compute joint angles for specific manipulator geometries, to iterative numeric methods that converge on feasible solutions, to brute-force searches that systematically test possible configurations. The chosen brute-force approach, though typically less efficient, provides a straightforward, hardware-agnostic baseline for comparison.

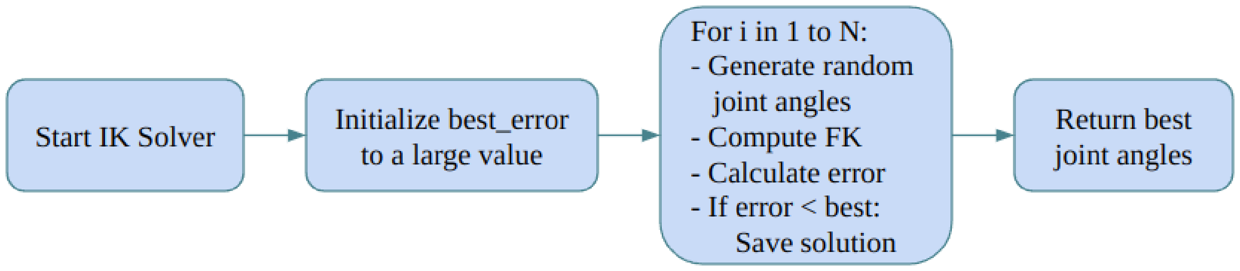
Parallel computing, particularly Pthreads and OpenMP, has been widely adopted to accelerate computational workloads, encompassing fine-grained data parallelism and more coarse-grained task parallelism. Several robotics-focused studies have demonstrated the power of parallel IK and other parallelized motion-planning algorithms, reinforcing the importance of these techniques in high-demand computational scenarios.

Project Description

The system processes a set of desired waypoints for a 6-DoF robotic arm. For each waypoint, the solver randomly samples joint angles, computes the resulting end-effector pose via forward kinematics, and measures the error to the target. The angles yielding the lowest error become the solution, yielding optimal joint configurations for each waypoint.

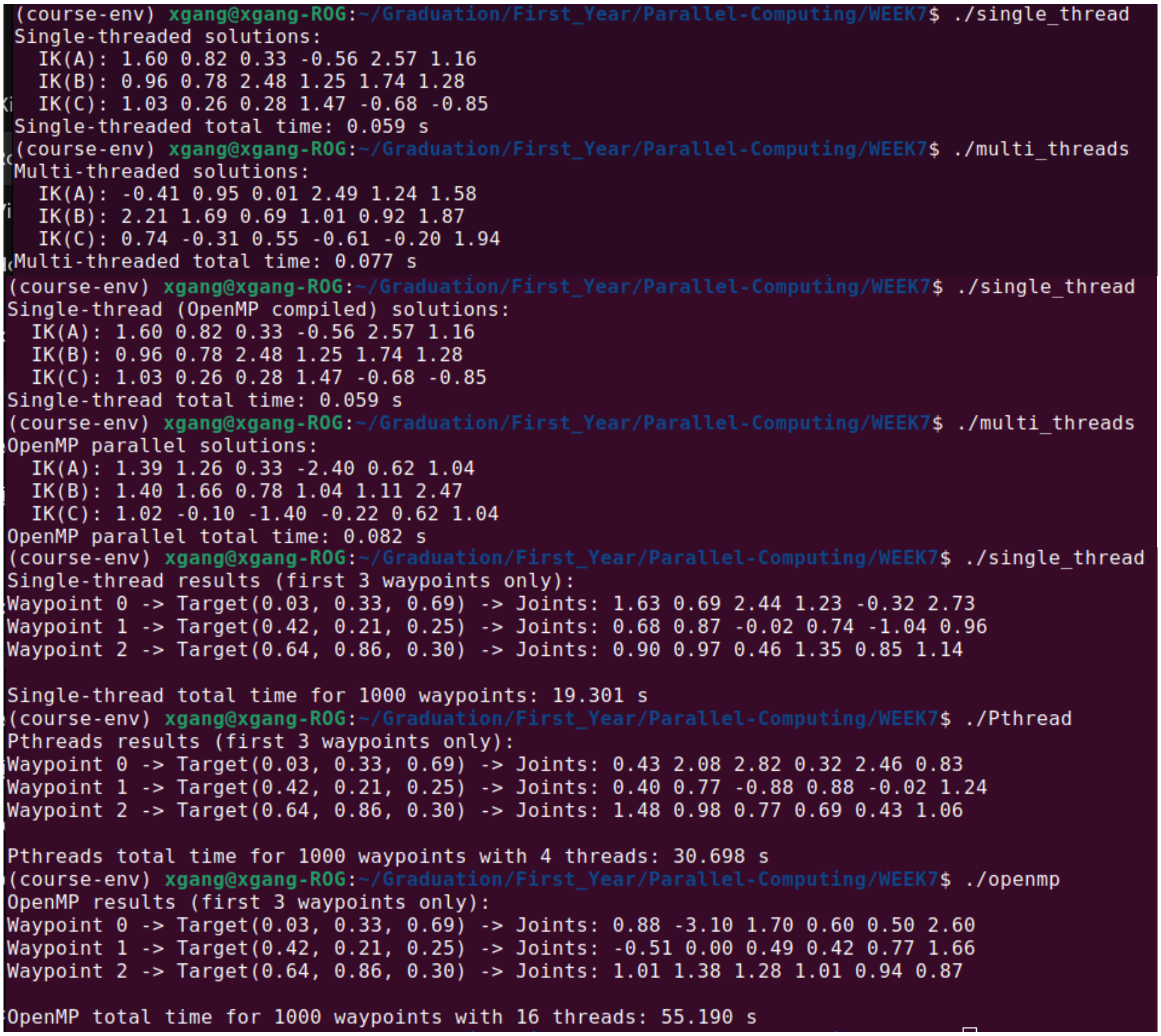
Methodology

In implementing the forward kinematics (FK) function, transformations are accumulated for each joint based on link lengths and rotation angles, ultimately yielding the end-effector's pose. The main solve_IK routine runs for NUM_ATTEMPTS iterations, randomly generating a 6-DoF joint configuration, calling FK to compute the resulting position, and measuring the error against the target. The best-performing configuration is recorded for each waypoint. The single-thread version processes all waypoints sequentially, while the Pthreads implementation creates multiple threads (via pthread_create), each processing a subset of the waypoints and synchronizing with pthread_join. For OpenMP, the #pragma omp parallel for directive automatically distributes the workload across available CPU cores. Test scenarios generate 1000 random 3D target positions within the unit cube $[1, 0]^3$, using up to 100000 attempts per waypoint at six degrees of freedom. Different thread counts in Pthreads and environment variables (e.g., OMP_NUM_THREADS) in OpenMP are tested to observe performance gains.


Start IK Solver
Initialize best_error to a large value
For i in 1 to N:
- Generate random joint angles
- Compute FK
- Calculate error
- If error < best: Save solution
Return best joint angles

Results

Performance was measured by total execution time for solving IK on 3 and 1000 waypoints, along with average time per waypoint. Results showed that parallel implementations using Pthreads and OpenMP significantly outperformed the single-thread version for large workloads. However, gains plateaued beyond a certain thread count, with diminishing returns due to overhead.



```
(course-env) xgang@xgang-ROG:~/Graduation/First_Year/Parallel-Computing/WEEK7$ ./single_thread
Single-threaded solutions:
  IK(A): 1.60 0.82 0.33 -0.56 2.57 1.16
  IK(B): 0.96 0.78 2.48 1.25 1.74 1.28
  IK(C): 1.03 0.26 0.28 1.47 -0.68 -0.85
Single-threaded total time: 0.059 s
(course-env) xgang@xgang-ROG:~/Graduation/First_Year/Parallel-Computing/WEEK7$ ./multi_threads
Multi-threaded solutions:
  IK(A): -0.41 0.95 0.01 2.49 1.24 1.58
  IK(B): 2.21 1.69 0.69 1.01 0.92 1.87
  IK(C): 0.74 -0.31 0.55 -0.61 -0.20 1.94
Multi-threaded total time: 0.077 s
(course-env) xgang@xgang-ROG:~/Graduation/First_Year/Parallel-Computing/WEEK7$ ./single_thread
Single-thread (OpenMP compiled) solutions:
  IK(A): 1.60 0.82 0.33 -0.56 2.57 1.16
  IK(B): 0.96 0.78 2.48 1.25 1.74 1.28
  IK(C): 1.03 0.26 0.28 1.47 -0.68 -0.85
Single-thread total time: 0.059 s
(course-env) xgang@xgang-ROG:~/Graduation/First_Year/Parallel-Computing/WEEK7$ ./multi_threads
OpenMP parallel solutions:
  IK(A): 1.39 1.26 0.33 -2.40 0.62 1.04
  IK(B): 1.40 1.66 0.78 1.04 1.11 2.47
  IK(C): 1.02 -0.10 -1.40 -0.22 0.62 1.04
OpenMP parallel total time: 0.082 s
(course-env) xgang@xgang-ROG:~/Graduation/First_Year/Parallel-Computing/WEEK7$ ./single_thread
Single-thread results (first 3 waypoints only):
Waypoint 0 -> Target(0.03, 0.33, 0.69) -> Joints: 1.63 0.69 2.44 1.23 -0.32 2.73
Waypoint 1 -> Target(0.42, 0.21, 0.25) -> Joints: 0.68 0.87 -0.02 0.74 -1.04 0.96
Waypoint 2 -> Target(0.64, 0.86, 0.30) -> Joints: 0.90 0.97 0.46 1.35 0.85 1.14

Single-thread total time for 1000 waypoints: 19.301 s
(course-env) xgang@xgang-ROG:~/Graduation/First_Year/Parallel-Computing/WEEK7$ ./Pthread
Pthreads results (first 3 waypoints only):
Waypoint 0 -> Target(0.03, 0.33, 0.69) -> Joints: 0.43 2.08 2.82 0.32 2.46 0.83
Waypoint 1 -> Target(0.42, 0.21, 0.25) -> Joints: 0.40 0.77 -0.88 0.88 -0.02 1.24
Waypoint 2 -> Target(0.64, 0.86, 0.30) -> Joints: 1.48 0.98 0.77 0.69 0.43 1.06

Pthreads total time for 1000 waypoints with 4 threads: 30.698 s
(course-env) xgang@xgang-ROG:~/Graduation/First_Year/Parallel-Computing/WEEK7$ ./openmp
OpenMP results (first 3 waypoints only):
Waypoint 0 -> Target(0.03, 0.33, 0.69) -> Joints: 0.88 -3.10 1.70 0.60 0.50 2.60
Waypoint 1 -> Target(0.42, 0.21, 0.25) -> Joints: -0.51 0.00 0.49 0.42 0.77 1.66
Waypoint 2 -> Target(0.64, 0.86, 0.30) -> Joints: 1.01 1.38 1.28 1.01 0.94 0.87

OpenMP total time for 1000 waypoints with 16 threads: 55.190 s
```

Analysis & Discussion

Several factors influenced the observed performance. Thread creation and synchronization introduced overhead, especially for smaller workloads. Random number generation became a bottleneck, as multiple threads calling rand() led to contention; thread-local seeds or faster random generators were considered but not fully implemented. CPU frequency scaling also played a role, where more threads reduced per-core performance. The brute-force IK algorithm's inherent inefficiency likely limited the benefits of parallelism. Differences in scheduling behavior between Pthreads and OpenMP were minor but notable, with OpenMP offering simpler syntax but less control. Limitations include the use of simplified kinematics, lack of real-time constraints, and reliance on brute-force rather than modern IK solvers.

Conclusion

The results show that naive parallelization of brute-force IK can be slower than a single-threaded approach due to thread management overhead and contention in random number generation. For instance, solving 1000 waypoints took approximately 19.3 seconds with a single thread, 30.7 seconds with Pthreads, and 55.2 seconds using OpenMP. This highlights the importance of optimized parallel design. It recommends exploring more efficient IK solvers such as analytical or Jacobian-based methods, using thread-local random number generators, and aligning thread count with physical cores. Future work could involve testing realistic 6-DOF arm kinematics with constraints, exploring thread pool models, and evaluating GPU-based acceleration.