AI1225 - Assignment 1 (Task 5)

Comparative Analysis: Manual vs. GenAI Implementation

Abror Shopulatov
AI1225 - Algorithms and Data Structures

February 13, 2026

1 Introduction

This document provides a comparative analysis between the solution implemented manually in Phase I and the solution generated by AI tools in Phase II for the Drone Collision Prevention assignment. The comparison focuses on algorithmic strategies, implementation details, code structure, and handling of advanced tasks.

2 Algorithmic Approach Comparison

2.1 Brute Force Algorithms

- **Phase I (Manual):** Implemented three distinct brute-force variations:
  1. Top-1 Basic Loop ($O(n^2)$).
  2. Top-k Naive (Store all n^2 distances and sort, $O(n^2 \log n)$).
  3. Top-k Optimized (Max-Heap of size k, $O(n^2 \log k)$).

  The manual approach explicitly explored the evolution from naive sorting to heap-based optimization within the brute-force paradigm.

- **Phase II (GenAI):** Implemented two variations:
  1. Top-1 Basic Loop ($O(n^2)$).
  2. Top-k utilizing a standard Max-Heap approach ($O(n^2 \log k)$).

  The AI skipped the intermediate "naive sort" step and immediately converged on the heap-optimized brute force solution as the standard baseline.

2.2 Optimized Algorithms (Divide and Conquer)

- **Phase I (Manual):** Focused on a pure Divide-and-Conquer (D&C) strategy for both Top-1 and Top-k. The Top-k logic relied on a custom implementation where the "strip" width was defined dynamically by the k-th smallest distance found so far. The recursion handled the merging of candidate lists directly.
- **Phase II (GenAI):** Adopted a bifurcated strategy:
  - **Top-1:** Used the classic Divide-and-Conquer algorithm ($O(n \log n)$).
  - **Top-k:** Switched to a **KD-Tree** based approach ($O(n \log n)$ average). The AI justified this switch by noting the implementation complexity of maintaining the "strip" property for $k > 1$ in pure D&C. The KD-Tree allows for efficient k-nearest neighbor queries without complex recursive merge steps.

3 Code Structure and Quality

Phase I (Manual)	**Phase II (GenAI)**
Modularity: Logic split into distinct functions within a main script or notebook context.	**Modularity:** Highly modularized with separate files (`utils.py`, `algorithm.py`, `main.py`, `live.py`).
Data Structures: Used standard Python lists and basic tuples.	**Data Structures:** Used typed `dataclasses` for Drones and Pairs, enhancing type safety and readability.
Libraries: Heavy reliance on NumPy for vectorized distance calculations.	**Libraries:** Used standard library (`math`) for core logic to demonstrate algorithmic purity, only using NumPy/Pandas for visualization/loading.
Error Handling: Basic input checking.	**Error Handling:** Robust correctness verification suites and edge case handling (e.g., $n < k$).

Table 1: Structural differences between implementations.

4 Task 4: Dynamic Updates

- **Phase I (Manual):** Proposed a theoretical incremental strategy: identifying moved points, invalidating their associated pairs in the top-k list, and recomputing distances only for those specific points. While logically sound, the implementation complexity of efficiently finding "affected pairs" without a spatial index is high.
- **Phase II (GenAI):** Implemented a concrete **Spatial Hash Grid**. This approach divides the coordinate space into buckets (cells). When a drone moves:

1. Its cell index is updated ($O(1)$). 2. Neighbors are queried only from adjacent cells. This provides a practical $O(1)$ update mechanism (average case) rather than re-scanning the list, offering a tangible speedup for dynamic simulations.

5 Summary of Findings

The manual Phase I submission demonstrated a strong grasp of the fundamental mechanics of Divide-and-Conquer, particularly in deriving the geometric constraints for the strip optimization. It correctly identified the complexity hierarchies ($O(n^2)$ vs $O(n^2 \log k)$ vs $O(n \log n)$).

The GenAI Phase II submission differed primarily in **engineering maturity** and **algorithm selection for extensions**. 1. **Engineering:** The AI code was structured as a production-ready package with typing, docstrings, and separate modules. 2. **Algorithmic Flexibility:** Instead of forcing D&C for Top-k, the AI selected the KD-Tree, which is standard in industry for this specific variant (k-NN search). 3. **Dynamic Solution:** The AI provided a concrete data structure (Spatial Grid) for Task 4, whereas Phase I focused on the logic of incremental updates without a specific spatial index.

Both approaches arrived at correct theoretical complexities and produced valid results, but Phase II leveraged a broader standard library of algorithms (KD-Trees, Spatial Hashing) to solve the extensions more efficiently.