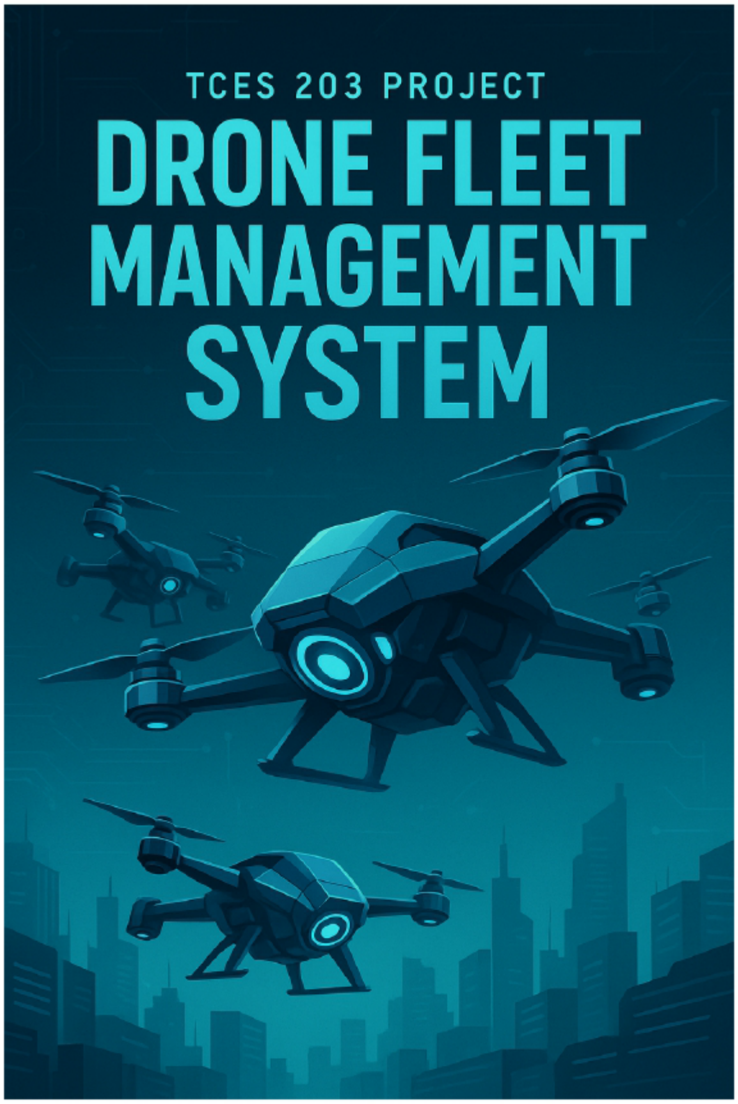
TCES 203 Programming Practicum – Autumn 2025

Instructor: Dr. Damiano Torre



Release Date: Monday, October 20, 2025 – 8:00 AM
Deadline: Sunday, October 26, 2025 – 11:59 PM
Points: 100 (50 Problem 1 + 50 Problem 2)
Work Mode: In team of two – one submission per team

Project Overview

Get ready to take off! In this project, you'll **build your own Drone Fleet Management System in C** — a fully functional program that tracks and manages a fleet of drones. Your system will store, search, and display drone information just like a real flight control dashboard.

But there's a twist: **you can only use one-dimensional and two-dimensional arrays** — no structs, pointers, or fancy data structures allowed (we did not cover that material.) Think of it as flying in manual mode: pure C fundamentals, full control! You're free to use **any loops or selection statements** (`for`, `while`, `do-while`, `if`, `switch`) to make your logic soar.

The project is divided into **two interconnected parts**, each worth **50%** of your total grade. Both must merge seamlessly into one clean, working program. You'll work **in pairs**, so plan your strategy, split your tasks wisely, and make sure your final submission **compiles, runs perfectly, and feels like one unified system**.

Your team's final code is what earns your grade — so make it solid, efficient, and ready to be deployed! 🚀

Problem 1 – Core Setup and Drone Management (50 points)

This part builds the foundation of your system. You will define data storage, handle basic input/output, and implement the core functions for adding and displaying drones.

Tasks

1. **Basic Setup (15 points)**
    - Define constants and macros for maximum fleet size and model name length.
    - Declare arrays for:
        - Drone IDs (`int ids[]`)
        - Model names (`char models[][MAX_LEN]`)
        - Battery levels (`float batteries[]`)
        - Positions (`float positions[][2]`)
2. **Adding Drones (15 points)**
    - Implement `addDrone()` to prompt for and store the drone's:
        - ID (positive integer)
        - Model name (string, up to `MAX_LEN`)
        - Battery level (float between 0.0 and 100.0)
        - Position (two floats: x and y coordinates)
    - Validate all inputs before storing.
3. **Displaying Drones (15 points)**
    - Implement `displayDrones()` to print all stored drones in a formatted table showing:
        - ID | Model | Battery | X | Y
    - Ensure proper column alignment and readable output.

Implement ONE Variation (for Problem 1)

You must implement one of the following variations (each worth the same total 5 points).
Your choice should be clearly commented at the top of your source file (e.g., `// Variation 1A: Basic Console Menu`).

Variation 1A – Basic Console Menu

- Create a simple menu that allows the user to:
    1. Add a drone
    2. Display all drones
    3. Exit
- Use a loop to keep showing the menu until "Exit" is chosen.

Variation 1B – Limited Capacity Simulation

- The fleet can store only up to 5 drones.
- When the user tries to add more, show a message:
  `"Fleet is full. Cannot add more drones."`
- Test by adding 6 drones to confirm correct behavior.

Variation 1C – Continuous Entry Mode

- Automatically enter drones in a loop until the user types a sentinel value (e.g., ID = -1).
- After exiting input mode, display all drones.

Problem 2 – Search, Calculations, and Robustness (50 points)

This part extends the system by adding search and analysis functions while maintaining the same array-based storage.
The functions in this part must **work with the data structures created in Problem 1**.

Tasks

1. **Search by ID (15 points)**
   - Implement `searchDroneByID(int id)` that looks up a drone by its ID.
   - If found, display all details.
   - If not found, print: `"Drone not found."`
2. **Average Battery Calculation (15 points)**
   - Implement `calculateAverageBattery()` to compute and display the average battery level of all drones.
   - If no drones exist, show `"No drones available."`
3. **Modularity and Input Validation (15 points)**
   - Create a user menu (or integrate with Problem 1's menu) that allows the user to choose:
     - Add a drone
     - Display all drones
     - Search by ID
     - Show average battery
     - Exit
   - Validate all user inputs and prevent invalid or out-of-range values.

Implement ONE Variation (for Problem 2)

Pick one of the following (each worth 5 points total for Problem 2).
Clearly comment your choice at the top of your file (e.g., `// Variation 2B: Nearest Drone Finder`).

Variation 2A – Enhanced Search

- Allow searching by either **ID** or **Model Name**.
- Make model name search case-insensitive (e.g., "Mavic" = "mavic").

Variation 2B – Nearest Drone Finder

- Add a function `findNearestDrone(float x, float y)` that finds and prints the drone closest to the provided position using Euclidean distance.
- You must not use any external libraries — compute distance manually using the standard formula.

Variation 2C – Battery Warning System

- After every `addDrone()` call, check for drones with a battery below 20%.
- Print a warning:
  `"Warning: Drone <ID> has low battery (<value>%)"`

Final Requirements

- The final submission must:
  - Compile and run without errors.
  - Be contained in **a single C file**.
  - Include **your names, and chosen variations** in a comment header.
  - Contain **clear comments** describing each function.
  - Use **only one-dimensional and two-dimensional arrays** for all stored data.
- **Submission:** Upload a single `.zip` file to Canvas containing your `.c` file.
- **No late submissions accepted**