Numerical Programming Final Project (Task 1)
Static Formation on a Handwritten Input

Student: **(Andria Gvaramia)** Course: Numerical Programming KIU

January 18, 2026

Task statement

Input: an image of a handwritten name (at least 8 characters), number of drones N, and a still initial configuration.
Goal: a drone swarm must move from the still initial formation and form the handwritten name.
Output: trajectory of each drone and visualization whose input is the trajectory.

1 Overview of the implemented pipeline

The implementation is split into three parts:

- **Target point extraction** from `task1/inputs/name.png`: produces N target points in pixel coordinates (x, y) saved as `task1/outputs/target_points.csv` and `task1/outputs/target_point`
- **Trajectory generation** (two alternative methods):
  1. **BVP via shooting** (per-drone boundary value targeting) in `task1/simulate_drones.py`.
  2. **Trajectory Optimization** (direct transcription + soft collision penalty) in `task1/optimal_trajec`
- **Visualization**: static trajectory plot and an animation (GIF or on-screen) showing drones moving toward the targets.

2 Input data and preprocessing

2.1 Binarization

The handwritten name image is loaded and binarized using OpenCV:

$$\text{binary}(x, y) = \begin{cases} 255 & \text{foreground (ink)} \\ 0 & \text{background} \end{cases}$$

using `THRESH_BINARY_INV`, so the ink becomes white (255). This makes subsequent distance transform and skeletonization operations straightforward.

2.2 Target points

The final targets are a set of N pixel coordinates:

$$T_i = (x_i^\star, y_i^\star), \quad i = 1, \ldots, N,$$

stored in `target_points.csv` (header: `x,y`) and `target_points.npy`.

Three extraction modes were used during development:

- **Contour sampling** (points on the boundary).
- **Interior sampling** using distance transform (points inside strokes, away from boundaries).
- **Skeleton (medial axis)** sampling via Zhang–Suen thinning (points lie on the centerline of the letters).

To reduce congestion and improve readability, a **minimum target spacing** can be enforced.

Optional debug figure. If generated, `debug_target_points.png` overlays the selected target points on the skeleton or mask.

3 Mathematical models

All models are implemented in 2D image coordinates (origin at top-left, y increases downward), consistent with plotting settings.

3.1 State definition

For drone i:

$$x_i(t) \in \mathbb{R}^2, \quad v_i(t) \in \mathbb{R}^2.$$

3.2 BVP model (via shooting)

The project slides also allow BVP formulations:

$$x_i(0) = x_{i,0}, \qquad x_i(T) = T_i.$$

In this project, a practical BVP solver is implemented using **shooting**:

- unknowns: initial velocity $v_i(0)$ for each drone,
- integrate the (single-drone) second-order ODE from $t = 0$ to $t = T$,
- adjust $v_i(0)$ (root finding / least squares) until $x_i(T) \approx T_i$.

Important limitation: the shooting implementation solves drones *independently* (no inter-drone coupling). Therefore it is excellent for hitting targets, but it does not guarantee collision-free motion.

3.3 Trajectory Optimization (direct transcription)

An alternative approach is implemented in `optimal_trajectories.py`. The optimization variables are the discrete positions

$$P_i^k \approx x_i(t_k), \quad k = 0, \ldots, K-1.$$

The objective contains:

- velocity smoothness: $\sum_k \|P^{k+1} - P^k\|^2$,
- acceleration smoothness: $\sum_k \|P^{k+2} - 2P^{k+1} + P^k\|^2$,
- goal term: $\|P^{K-1} - T\|^2$,
- collision penalty (soft constraint): hinge-squared on pairwise distances smaller than R_{safe}.

The solver uses gradient descent on this objective.

4 Numerical methods

4.1 Shooting for BVP

For each drone, a nonlinear system is solved in the unknown initial velocity. Two modes exist:

- **Position-only**: solve $F(v_0) = x(T; v_0) - T = 0$ via root finding.
- **Position + final-velocity damping**: solve in least-squares sense using residual

$$r(v_0) = \big[x(T; v_0) - T,\ w_v\, v(T; v_0)\big],$$

  which reduces end oscillations by encouraging $v(T) \approx 0$.

4.2 Trajectory optimization

The optimization uses fixed-step gradient descent with learning rate `--lr` and number of iterations `--iters`. Collision penalties can be computed every s frames (`--collision-stride`) to trade accuracy for speed.

5 Initialization and assignment

5.1 Initial formations

The initial configuration is a still formation such as:

- a horizontal line below the name (`hline_below`),
- a horizontal line matching each target's x coordinate (`hline_match_targets_x`),
- two horizontal lines below (`two_hlines_below`).

Minimum spacing between drones in the initial formation is enforced to prevent immediate collisions.

5.2 Drone-to-target assignment

In the implemented Task 1 setup, drone i is assigned to target T_i (row-wise indexing in `target_points.csv`). During development, sorting target points (e.g., by x or by y) was used to reduce crossing trajectories.

6 Reproducibility (commands)

All experiments are reproducible from the project root.

6.1 Extract targets (example)

```
python3 task1/extract_target_points.py --n 100 --mode skeleton \
  --min-target-spacing 5 --debug-png
```

6.2 BVP shooting (accurate target hitting)

```
python3 task1/simulate_drones.py \
  --bvp-match-final-velocity --bvp-final-velocity-weight 3.0 \
  --targets task1/outputs/target_points.csv --bg-image task1/inputs/name.png \
  --initial hline_below --offset 50 \
  --k-p 2.0 --k-d 2.5 --t-end 20 --steps 200 --show
```

6.3 Trajectory Optimization (direct transcription)

```
python3 task1/optimal_trajectories.py \
  --targets task1/outputs/target_points.csv --bg-image task1/inputs/name.png \
  --initial hline_below --offset 50 --init-min-spacing 25 \
  --k-steps 80 --iters 80 --lr 1e-4 \
  --r-safe 10 --w-col 50 --show
```

7 Validation and test cases

7.1 Metrics

Two practical metrics were used:

- **Target accuracy**: $\|x_i(T) - T_i\|$ (mean/max over drones).
- **Safety**: minimum pairwise distance $\min_{i<j,t} \|x_i(t) - x_j(t)\|$ and/or collision count for a chosen threshold.

7.2 Expected behavior

- **BVP shooting** should hit targets very well, but may allow path crossings because inter-drone coupling is not modeled.
- **Trajectory Optimization** can trade off smoothness vs. collision avoidance via weights; it is slower but often yields more controlled behavior.

7.3 Limitations

- **BVP shooting limitation**: independent drones $\Rightarrow$ no collision-free guarantee.
- **Trajectory Optimization**: sensitive to hyperparameters (learning rate, weights); may require tuning per scene.

8 AI usage disclosure

This project was developed with AI assistance (ChatGPT) used for:

- explaining BVP and shooting concepts,
- proposing model/parameter adjustments,
- helping implement and debug Python code, CLI arguments, and visualization.

All final code and results were verified by running the scripts locally and inspecting outputs.

Files included in the submission

- Code: `task1/extract_target_points.py`, `task1/simulate_drones.py`, `task1/optimal_trajectorie` `task1/sort_target_points.py`
- Input: `task1/inputs/name.png`
- Generated data (reproducible): files in `task1/outputs/` (targets, trajectories, plots, animations)
- This report: `task1/report/main.tex`