Numerical Programming Final Project
Combined Presentation (Tasks 1–3)

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

January 23, 2026

Problem statement

Simulate an illuminated drone show with shape preservation:

1. Static formation on a handwritten input.
2. Transition to "Happy New Year!" greeting.
3. Dynamic tracking of a moving object from video.

1 Task 1: Static Formation on a Handwritten Input

Task statement

Input: handwritten name image (at least 8 characters), number of drones N, still initial configuration.
Goal: move the swarm from the still initial formation and form the handwritten name.
Output: trajectories and visualization.

1.1 Overview of the pipeline

- **Target point extraction** from `task1/inputs/name.png` into `task1/outputs/target_points.csv`.
- **Trajectory generation** using swarm IVP with repulsion in `task1/simulate_drones.py`.
- **Visualization**: static trajectory plot and an animation (GIF).

1.2 Input data and preprocessing

Binarization uses `THRESH_BINARY_INV` so ink becomes white and background black. Targets are sampled from contour / interior / skeleton, with optional minimum spacing.

1.3 Mathematical model

For drone i:

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

Swarm IVP with repulsion:

$$\dot{x}_i(t) = v_i(t), \tag{1}$$

$$\dot{v}_i(t) = \frac{1}{m}\left(k_p\,(T_i - x_i(t)) + \sum_{j\neq i} f_{\text{rep}}(x_i, x_j) - k_d\, v_i(t)\right). \tag{2}$$

BVP (shooting, optional):

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

1.4 Numerical methods

We use `solve_ivp` (RK45) for the coupled IVP. BVP shooting is available per drone without repulsion.

1.5 Reproducibility (commands)

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

```
python3 task1/simulate_drones.py \
  --model swarm --k-rep 200 --r-safe 12 \
  --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 --v-max 1e9 --t-end 20 --steps 200 --show
```

1.6 Test cases

Works well: clear, high-contrast handwriting; moderate N with spacing; tuned R_{safe}.
Does not work well: low-contrast inputs; very large N with small R_{safe}; shooting without repulsion.

2 Task 2: Transition to "Happy New Year!"

Task statement

Input: swarm at Task 1 formation and greeting text.
Goal: move from handwritten name to greeting formation.
Output: trajectories and visualization.

2.1 Overview

Start positions are `task1/outputs/target_points.csv`. Greeting targets are extracted via `extract_target_points.py`. Transition trajectories are generated in `task2/transition.py` using BVP shooting or swarm IVP.

2.2 Mathematical model

Same second-order point-mass model as Task 1 with fixed targets T_i for the greeting.

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

2.3 Reproducibility (commands)

```
python3 task2/generate_greeting_image.py --out task2/inputs/greeting.png
```

```
python3 extract_target_points.py \
  --image task2/inputs/greeting.png \
  --n 100 --mode skeleton --min-target-spacing 5 \
  --out-dir task2/outputs --debug-png
```

```
python3 task2/transition.py \
  --start task1/outputs/target_points.csv \
  --targets task2/outputs/target_points.csv \
  --bg-target task2/inputs/greeting.png \
  --model swarm --k-rep 200 --r-safe 12 \
  --k-p 2.0 --k-d 2.5 --v-max 1e9 \
  --t-end 20 --steps 200 \
  --collision-report --collision-threshold 12 \
  --save-gif --save-traj-csv --save-traj-npy --save-traj-plot
```

2.4 Test cases

Works well: same N for start/target; swarm IVP with repulsion; skeleton extraction for greeting.
Does not work well: mismatched N; shooting without repulsion in dense cases; very small R_{safe}.

3 Task 3: Dynamic Tracking and Shape Preservation

Task statement

Input: swarm at Task 2 greeting and a video.
Goal: track a moving object with shape preservation.
Output: trajectories and visualization.

3.1 Overview

We segment the moving object, extract its contour per frame, sample N boundary points, and maintain stable correspondence across frames. The implementation is in `task3/dynamic_tracking.py`.

3.2 Contour targets

For each frame k, sample N points at equal arc-length:

$$T_i^{(k)} = C_k(s_i), \quad s_i = \frac{i}{N} L(C_k).$$

We align consecutive frames via cyclic shift and orientation checks to keep point correspondence stable.

3.3 Controllers

- **Direct (kinematic)**: $x_i^{(k)} = T_i^{(k)}$ (zero tracking error).
- **Dynamics (IVP RK4)**: second-order damped model integrated by RK4 to follow moving targets.

3.4 Reproducibility (commands)

```
python3 task3/dynamic_tracking.py \
  --tracking-mode contour \
  --controller direct \
  --video-step 1 \
  --contour-upscale 3.0 --contour-smooth 9 \
  --segmenter greenscreen \
  --save-gif --save-traj-csv --save-traj-npy --gif-fps 30 \
  --output-gif task3/outputs/task3_contour_direct_hi_with_video.gif
```

3.5 Test cases

Works well: green-screen video; direct controller; higher contour upscaling.
Does not work well: complex backgrounds; dynamics controller at high speed (lag); very small N.

AI usage disclosure

This project was developed with AI assistance (ChatGPT) for explanation, implementation, and debugging support.

Files included in the submission

- Code: `extract_target_points.py`, `task1/simulate_drones.py`, `task2/transition.py`, `task3/dynamic_tracking.py`
- Inputs: `task1/inputs/name.png`, `task2/inputs/greeting.png`, `task3/video.mp4`
- Outputs: `task1/outputs/`, `task2/outputs/`, `task3/outputs/`
- Presentation (this file): `report/presentation.pdf`