Subtask 3: Dynamic Tracking and Shape Preservation via Optical Flow

Numerical Programming Final Project (Illuminated Drone Show Simulation)

Student: <YOUR NAME> Instructor: Ramaz Botchorishvili

Abstract

Subtask 3 requires a drone swarm to transition from a static greeting formation to a *moving* object in a reference video and to reproduce its motion over time while maintaining a coherent swarm shape. This report presents a requirements-aligned formulation as an Initial Value Problem with Velocity Tracking (IVP-VT), where a dense optical-flow field provides a time-varying velocity command $V(x, t)$ extracted from the video. The swarm dynamics combine (i) attraction to per-frame targets, (ii) velocity matching to the local video motion, (iii) collision avoidance via repulsive forces, and (iv) velocity saturation. A fourth-order Runge–Kutta method (RK4) integrates the coupled ODE system. We provide a full pipeline from 2D video motion to (optionally) 3D swarm trajectories, code–math traceability, and verification metrics (tracking error, velocity error, minimum separation, and shape preservation).

Contents

1 Requirement Traceability and Problem Definition

1.1 Project requirement (Subtask 3)

Given an initial swarm configuration already forming the "Happy New Year!" greeting, and a user-selected reference video, the task is:

- transition from the greeting formation to a moving object/structure in the video,
- **dynamically repeat the object's motion** over time,
- maintain **shape preservation** and **collision-free** trajectories,
- output the full trajectories and visualization (illumination maintained throughout).

1.2 Inputs and outputs

Inputs:

- Swarm state at end of Subtask 2: $\{x_i(t_0), v_i(t_0)\}_{i=1}^{N}$.
- Reference video frames $\{I(\cdot,\cdot,t_k)\}_{k=0}^{K}$ (grayscale).
- Number of drones N, physical parameters $(m, v_{\max}, k_p, k_v, k_d, k_{\text{rep}}, R_{\text{safe}})$.

Outputs:

- Trajectories $\{x_i(t)\}_{i=1}^{N}$ for all time steps.
- Rendered animation/video of the illuminated swarm following the reference motion.
- Verification statistics: tracking error, separation margin, and velocity constraint checks.

1.3 Why IVP-VT?

Because the swarm begins from a *known* initial state and must evolve under time-varying guidance extracted from video, the natural formulation is an **Initial Value Problem with Velocity Tracking (IVP-VT)**:

$$x_i(t_0) = x_{i,0}, \quad v_i(t_0) = v_{i,0}, \quad \text{and } (x_i(t), v_i(t)) \text{ follow an ODE driven by } V(x,t).$$

2 Mathematical Model

2.1 State variables and dimension

Each drone $i \in \{1, \ldots, N\}$ has position $x_i(t) \in \mathbb{R}^d$ and velocity $v_i(t) \in \mathbb{R}^d$, where $d = 2$ (planar motion) or $d = 3$ (with altitude z). The stacked state is

$$Y(t) = (x_1, v_1, \ldots, x_N, v_N) \in \mathbb{R}^{2dN}.$$

2.2 Velocity saturation

Physical drones obey a speed limit. For any vector $u \in \mathbb{R}^d$, define the saturated version

$$\text{sat}(u) = u \cdot \min\left(1, \frac{v_{\max}}{\|u\|}\right), \qquad \text{with sat}(0) = 0. \tag{1}$$

2.3 Collision avoidance (repulsive force)

For safety radius $R_{\text{safe}} > 0$, a common repulsion model is

$$f_{\text{rep}}(x_i, x_j) = \begin{cases} k_{\text{rep}} \dfrac{x_i - x_j}{\|x_i - x_j\|^3}, & \|x_i - x_j\| < R_{\text{safe}}, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The total repulsion on drone i is $\sum_{j \neq i} f_{\text{rep}}(x_i, x_j)$.

2.4 Time-varying target points from video

Let $T_i(t) \in \mathbb{R}^d$ denote the time-varying target for drone i. In practice, $T_i(t_k)$ is obtained by:

1. extracting a foreground/edge set from frame $I(\cdot, \cdot, t_k)$,
2. sampling exactly N points,
3. assigning drones to sampled points (greedy or Hungarian) to minimize travel and preserve continuity.

This yields a discrete target sequence $\{T_i(t_k)\}_{k=0}^{K}$.

2.5 IVP-VT governing equations

Subtask 3 augments static tracking with *velocity matching* to the video motion. The IVP-VT model is:

$$\dot{x}_i(t) = \text{sat}(v_i(t)), \tag{3}$$

$$\dot{v}_i(t) = \frac{1}{m}\Big[k_p\big(T_i(t) - x_i(t)\big) + k_v\big(V_{\text{sat}}(x_i(t), t) - v_i(t)\big) - k_d v_i(t) + \sum_{j \neq i} f_{\text{rep}}\big(x_i(t), x_j(t)\big)\Big]. \tag{4}$$

The velocity field $V(x, t)$ is extracted from optical flow (Section **??**), and its saturated version is

$$V_{\text{sat}}(x, t) = \text{sat}(V(x, t)), \tag{5}$$

ensuring $\|V_{\text{sat}}(x, t)\| \leq v_{\max}$.

2.6 Interpretation of terms

Equation (**??**) decomposes the acceleration into:

- **Spring attraction** $k_p(T_i - x_i)$: pulls drones onto the instantaneous target shape.
- **Velocity matching** $k_v(V_{\text{sat}} - v_i)$: makes drones *follow video motion* rather than lagging behind positions only.
- **Damping** $-k_d v_i$: removes energy, suppresses oscillation and stabilizes tracking.
- **Repulsion** $\sum f_{\text{rep}}$: ensures collision avoidance and local shape spacing.

3 From 2D Video Motion to 3D Swarm Motion

3.1 2D projection space

Video frames define a 2D image plane with pixel coordinates (x, y). All extracted points and optical flow live in this plane.

3.2 Embedding into 3D

To exploit additional degrees of freedom, the swarm can be simulated in $d = 3$:

$$x_i(t) = (x_i^{(x)}(t), x_i^{(y)}(t), x_i^{(z)}(t)).$$

A practical choice is to:

- keep targets in the image plane: $T_i(t_k) = (T_{i,x}(t_k), T_{i,y}(t_k), z_i^\star)$,
- assign a fixed or slowly varying altitude layer $z_i^\star$ to distribute drones in depth,
- set the video-derived velocity to have zero vertical component: $V(x,t) = (u(x,y,t), v(x,y,t), 0)$.

This maintains faithful 2D shape reproduction while improving collision flexibility in 3D.

4 Optical Flow and Velocity Field Extraction

4.1 Brightness constancy and optical flow constraint

Optical flow assumes *brightness constancy*:

$$I(x,y,t) \approx I(x+u,\, y+v,\, t+\Delta t),$$

where (u, v) is the apparent pixel displacement. Linearizing (first-order Taylor expansion) yields the optical-flow constraint:

$$I_x u + I_y v + I_t = 0, \tag{6}$$

with spatial gradients I_x, I_y and temporal gradient I_t.

4.2 Lucas–Kanade local least squares

Equation (**??**) is underdetermined at a single pixel. Lucas–Kanade solves it over a local window Ω around each pixel by minimizing

$$\min_{u,v} \sum_{(x,y)\in\Omega} \big(I_x u + I_y v + I_t\big)^2.$$

This yields a 2×2 normal equation:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}. \tag{7}$$

Solving (**??**) for each pixel yields a dense flow field $V(x,y,t) = (u(x,y,t), v(x,y,t))$.

4.3 From flow to swarm guidance

To guide drone i at position $x_i(t)$, the flow is sampled at its image-plane coordinates:

$$V(x_i(t), t) = \text{bilinear\_sample}\big(\text{flow}(\cdot,\cdot,t),\, x_i(t)\big).$$

Bilinear interpolation avoids discontinuities in the velocity command.

4.4 Saturation and scaling

Since optical flow can contain outliers, the pipeline applies:

- saturation V_{sat} as in (**??**),
- optional scaling to match simulation units (pixels/sec).

5 Numerical Method and Discretization

5.1 Time discretization

Let video frames be indexed by $k = 0, \ldots, K$ with frame interval Δt_{vid}. The simulation uses a smaller integration step h (e.g., $h = 0.05\,\text{s}$) and performs S RK4 steps per video frame:

$$\Delta t_{\text{vid}} \approx S\, h.$$

This decouples visualization rate from integration accuracy.

5.2 RK4 integration

For a general ODE $\dot{y} = f(t, y)$, RK4 updates

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

with

$$k_1 = f(t_n, y_n), \quad k_2 = f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_1), \quad k_3 = f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_2), \quad k_4 = f(t_n + h, y_n + hk_3).$$

In our case, y stacks all drone positions and velocities, and f implements (**??**)–(**??**).

5.3 Stability considerations

The system is a damped, driven second-order model with time-varying forcing. Practical stability is achieved by:

- choosing k_d sufficiently large to avoid oscillation,
- keeping k_v moderate to prevent "over-chasing" noisy flow,
- limiting speeds via saturation (**??**) and (**??**),
- using a sufficiently small h for numerical stability under strong repulsion.

6 Algorithmic Pipeline (End-to-End)

For each video frame k:

1. **Preprocess**: read frame $I(\cdot, \cdot, t_k)$ and convert to grayscale.
2. **Targets**: extract edges/foreground and sample N points $\{T_i(t_k)\}$.
3. **Assignment**: match drones to targets to minimize distance and preserve continuity.
4. **Optical flow**: compute dense flow between consecutive frames to obtain $V(\cdot, \cdot, t_k)$.
5. **Sample flow**: evaluate $V(x_i(t), t)$ at each drone position via bilinear interpolation.
6. **Integrate**: run S RK4 steps for (**??**)–(**??**) to advance the swarm.
7. **Render**: store positions for visualization; repeat.

7 Selected Code Segments and Mathematical Meaning

This section provides traceability between the mathematical model and the implementation in `drone_show_math.py`.

7.1 IVP-VT derivative (acceleration terms)

The acceleration in (**??**) is implemented as a sum of spring, velocity matching, damping, and repulsion.

Listing 1: Acceleration composition matches Eq. (**??**).

```
spring_force = K_P * (self.tgt - p)            # k_p (T - x)
if self.use_velocity_matching:
    velocity_match_force = K_V * (self.tgt_vel - v)  # k_v (V - v)
else:
    velocity_match_force = 0
damping = K_D * v                              # k_d v
dv = (spring_force + velocity_match_force + rep - damping) / M
```

7.2 Velocity saturation

Equation (**??**) uses $\dot{x} = \mathrm{sat}(v)$, implemented as:

Listing 2: Velocity saturation implements Eq. (**??**).

```
vnorm = np.sqrt(np.sum(v**2, axis=1, keepdims=True))
dx = v * np.minimum(1.0, V_MAX / np.maximum(vnorm, 1e-6))
```

7.3 Repulsive force

The repulsive force in (**??**) is computed for neighbor pairs within R_{safe} using spatial hashing (sub-quadratic neighbor search):

Listing 3: Repulsion implements Eq. (**??**).

```
diff = self.pos[pairs[:,0]] - self.pos[pairs[:,1]]
dist = np.sqrt(np.sum(diff**2, axis=1))
fvec = (K_REP / np.maximum(dist, 0.1)**3)[:,None] * diff
np.add.at(forces, pairs[:,0],  fvec)
np.add.at(forces, pairs[:,1], -fvec)
```

7.4 Optical flow to velocity field

The code computes dense flow (Lucas–Kanade style) and returns a field `flow[h,w,2]` corresponding to (u, v):

Listing 4: Optical flow returns a dense velocity field $V(x, y, t)$.

```
Ix = convolve_2d(curr, sobel_x)
Iy = convolve_2d(curr, sobel_y)
It = curr - prev
# window sums -> solve 2x2 system per pixel (Lucas-Kanade)
det = Ix2 * Iy2 - IxIy * IxIy
flow[:,:,0] = np.where(valid, (Iy2 * (-IxIt) - IxIy * (-IyIt)) / (det +
    1e-10), 0)
flow[:,:,1] = np.where(valid, (Ix2 * (-IyIt) - IxIy * (-IxIt)) / (det +
    1e-10), 0)
```

7.5 Sampling the velocity field

Bilinear interpolation samples $V(x_i, t)$ continuously from the pixel grid:

Listing 5: Bilinear sampling computes $V(x_i, t)$ for each drone.

```
f00 = flow[y0, x0]; f01 = flow[y0, x1]
f10 = flow[y1, x0]; f11 = flow[y1, x1]
vel_2d = (f00*(1-fx)[:,None]*(1-fy)[:,None] +
          f01*fx[:,None]*(1-fy)[:,None] +
          f10*(1-fx)[:,None]*fy[:,None] +
          f11*fx[:,None]*fy[:,None])
```

7.6 Target-velocity estimate

The implementation also supports an equivalent velocity-tracking term via finite differences:

$$\dot{T}(t_k) \approx \frac{T(t_k) - T(t_{k-1})}{\Delta t_{\text{vid}}}.$$

Listing 6: Finite-difference target velocity for velocity matching.

```
self.tgt_vel = (new_tgt - self.prev_tgt) / (frame_dt * steps * DT)
```

8 Verification, Validation, and Test Cases

8.1 Core quantitative checks

We verify the simulation by measuring:

(1) Position tracking error

$$e_x(t) = \frac{1}{N} \sum_{i=1}^{N} \|x_i(t) - T_i(t)\|.$$

A decreasing or bounded $e_x(t)$ indicates successful tracking.

(2) Velocity tracking error

$$e_v(t) = \frac{1}{N} \sum_{i=1}^{N} \|v_i(t) - V_{\text{sat}}(x_i(t), t)\|.$$

A small e_v indicates the swarm follows video motion.

(3) Collision margin

$$d_{\min}(t) = \min_{i \neq j} \|x_i(t) - x_j(t)\|.$$

We require $d_{\min}(t) \geq R_{\text{safe}}$ (or at least no sustained violations).

(4) Speed constraint

$$\|\dot{x}_i(t)\| \leq v_{\max} \quad \forall i, t,$$

guaranteed by saturation, but still tracked to detect numerical issues.

8.2 Shape preservation metrics

To demonstrate "shape preservation" beyond collision avoidance, we evaluate:

(A) Centroid and scale consistency Let $c(t) = \frac{1}{N}\sum_i x_i(t)$ and normalized positions $\tilde{x}_i(t) = x_i(t) - c(t)$. We monitor $\|c(t) - c_T(t)\|$ and the swarm radius

$$r(t) = \sqrt{\frac{1}{N}\sum_i \|\tilde{x}_i(t)\|^2},$$

to ensure the swarm does not collapse or explode.

(B) Chamfer/Hausdorff-style distance (optional) Given the sampled target set $mathcalT(t)$ and swarm set
$mathcalX(t)$, a symmetric Chamfer distance

$$d_C(mathcalX, mathcalT) = \frac{1}{|mathcalX|}\sum_{x\in mathcalX}\min_{T\in mathcalT}\|x-T\| + \frac{1}{|mathcalT|}\sum_{T\in mathcalT}\min_{x\in mathcalX}\|T-x$$

quantifies shape similarity even under permutation.

8.3 Test cases

Recommended tests:

- **Slow motion video**: flow is small; tracking should be accurate and stable.
- **Fast motion video**: tests saturation and responsiveness (k_v).
- **Low-texture video**: flow may be unreliable; k_v should be reduced, and targets dominate.
- **Crowded shape**: high density tests repulsion and neighbor search robustness.

9 Complexity and Performance Considerations

9.1 Computational bottlenecks

Major costs per frame:

- Optical flow: dense per-pixel operations, $\mathcal{O}(HW)$.
- Assignment: greedy $\mathcal{O}(N^2)$ (acceptable for $N \approx 2000$); Hungarian is $\mathcal{O}(N^3)$.
- Repulsion: accelerated neighbor search via spatial hashing approaches $\mathcal{O}(N)$ expected.
- RK4 integration: $\mathcal{O}(SN)$ per frame (times force evaluations).

9.2 Practical tuning guidelines

- Increase k_p to tighten formation to targets, but too large can cause stiffness.
- Increase k_v to better follow motion; reduce it for noisy flow to avoid jitter.
- Increase k_d to reduce oscillations; overly large k_d can slow response.
- Adjust R_{safe} and k_{rep} to balance safety vs. formation fidelity.
- If $d = 3$, distribute altitude layers to reduce collision probability during fast motion.

10 Task–Solution Mapping Table (Requirements-First)

Requirement / Task	Implemented Solution	Mathematical Basis	Code Reference
Transition from greeting to moving object	Per-frame targets from video frames	$T_i(t)$ time-varying targets	`load_animation()`, `extract_points()`
Dynamic motion replication	Optical-flow velocity field sampled at drones	$V(x,t)$, $V_{\text{sat}}(x,t)$	`compute_optical_flow()`, `sample_velocity_field()`
Velocity tracking	Add velocity matching term	$k_v(V_{\text{sat}} - v)$ in (**??**)	`K_V*(tgt_vel - v)`
Shape preservation	Continuous reassignment + damping + repulsion	stable driven dynamics, permutation-invariant metrics	`greedy_assignment()`, `repulsion()`
Collision avoidance	Repulsive force within R_{safe}	Eq. (**??**)	`find_neighbor_pairs_3d()`, `repulsion()`
Speed limit	Saturate velocities	Eq. (**??**)–(**??**)	velocity scaling in `deriv()`
Numerical solver	RK4 integration per frame	4th-order IVP method	`Swarm.step()`
Illuminated visualization	Render all frames with visible markers	visualization requirement	`render_video()`

11 Limitations and Failure Modes

- **Optical flow ambiguity**: low-texture regions violate assumptions behind (**??**); the flow can be noisy or zero.
- **Large displacements**: for fast motion, small-window Lucas–Kanade can underestimate; multi-scale flow would improve robustness.
- **Assignment jitter**: reassigning per frame can introduce discontinuities if targets change abruptly; temporal smoothing or multi-frame assignment can reduce flicker.
- **Unit mismatch**: video flow is in pixels/frame; careful scaling to pixels/second is required for physically plausible gains.

12 AI Usage Statement

AI assistance was used for report structuring, mathematical exposition, and consistency checks between the project requirements, derived equations, and the provided implementation. All algorithmic decisions (model choice, parameters, numerical method) and final code execution/outputs are based on the student's implementation.

13 Conclusion

Subtask 3 was formulated as an IVP with Velocity Tracking (IVP-VT) driven by an optical-flow-derived velocity field. The model combines attraction to time-varying targets, velocity matching to reproduce video motion, repulsive collision avoidance, damping, and strict velocity saturation. An RK4 integrator provides stable time stepping under stiff repulsion and time-varying forcing. The presented pipeline is requirements-aligned and verifiable through tracking, safety, and shape-preservation metrics, providing a complete dynamic tracking solution for the illuminated drone show simulation.