# Assessment 5 : CCCS660 Computational Intelligence

Quentin Lao — Matricule : 261233739

## HANDS-ON OPTION

This assessment focuses on the interaction between autonomous drones and a human-controlled leader drone for creating orthomosaics.

## 1 Modeling of a drone movement

In this section, we will specify our approach to modeling the movement controller for a drone. This describes as well the method of movement control we implemented in our Python code.

We model the movement of a drone as a flow of *movement signals* received at intervals of $\Delta t$ milliseconds. Each *movement signals* is represented by a 4-uplet $(\ell, r, u, d)$ where each component is either 0 or 1 and corresponds to a specific direction ($\ell$ for left, $r$ for right, $u$ for up and $d$ for down). Thus, every $\Delta t$ milliseconds, a drone receives a quadruple telling it what direction to take. Because of physical inertia, the drone's speed is managed by assigning non-zero values to certain signals. For instance, a flow of signals $(0,1,0,0) \rightarrow (0,1,0,0) \rightarrow (0,1,0,0) \rightarrow (0,1,0,0) \rightarrow (0,1,0,0)$ instructs the drone to move right at maximum speed, whereas the flow $(0,1,0,0) \rightarrow (0,0,0,0) \rightarrow (0,1,0,0) \rightarrow (0,0,0,0) \rightarrow (0,1,0,0)$ is a slower movement on the right.

## 2 Drone formation for our problem

To create orthomosaics, it is crucial for the drones to maintain a cohesive and uniform group movement. This means that the autonomous drones should as much as possible preserve their relative position with the leader drone : if the leader drone moves right, all autonomous drones should also move right.

All drones should fly at the same altitude to make sure that the images have an uniform scale. We propose to use a V-shaped formation. The V-shape allows for a wide coverage of the area and minimizes image overlap. In addition, it reduces the risk of collisions as it prevents drones from coming into direct paths with each other. An example of this formation is used by the "Patrouille de France" in the French Army : https://www.youtube.com/watch?v=AK-gYfsIaCc.

Figure 1 illustrates the V-shape formation for our 6 drones.

## 3 Relative position and movement rules

Initially at time $t = 0$, the drones are positioned in the V-shape formation. Each autonomous drone $i \in \{1, ..., 5\}$ starts at a relative position $(\Delta x_i(0), \Delta y_i(0))$ from the leader drone which is at the absolute coordinates $(x_0(0), y_0(0))$. The goal is to preserve as much as possible these relative positions of the autonomous drones to the leader drone.

The leader drone broadcasts its location and velocity in all direction every $\delta$ milliseconds through a wave signal that propagates in the air. Thanks to Doppler effect [1], autonomous drones can determine their

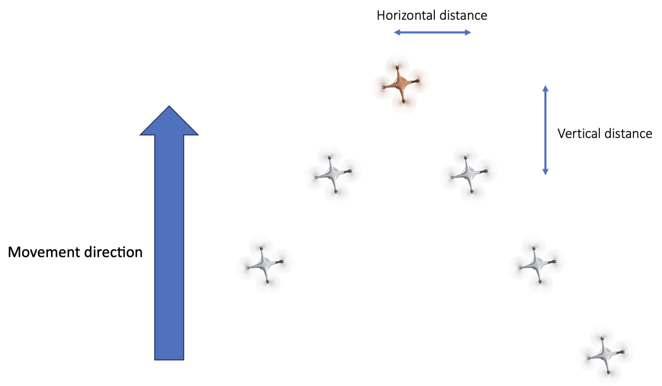


FIGURE 1 – V-shape formation for our drones (the red drone in the leader one)

relative positions to the leader drone. Here is the movement rule when at time $t$, an autonomous drone $i$ has calculated its relative positions $(\Delta x_i(t), \Delta y_i(t))$ after receiving a signal from the leader drone :

1. If $\Delta x_i(t) > \Delta x_i(0)$, the drone is too far to the right and should move left. It then sets the left movement component $\ell = 1$. Otherwise, it should move right, setting $r = 1$. An example of this is shown in figure 2a.
2. Similarly for the vertical position, the drone should move up $(u = 1)$ or down $(d = 1)$ to return to its original relative vertical position. An example of this is shown in figure 2b.
3. This forms a new *movement signal* for the autonomous drone. This is illustrated in Figure 2c. This *movement signal* is repeated until the next update from the leader drone is received.

In practice with our Python implementation, we noticed an oscillation issue : if $\Delta x_i(t) > \Delta x_i(0)$, the drone moves left but it could exceed the target position to the extent that $\Delta x_i(t) < \Delta x_i(0)$ in the following update causing the drone to move right, and so on. To address this problem, we introduce a *security zone* $\varepsilon > 0$. Now, the drone moves left only if $\Delta x_i(t) > \Delta x_i(0) + \varepsilon$, moves right only if $\Delta x_i(t) < \Delta x_i(0) - \varepsilon$ and stop moving otherwise [1].

## 4 Collision Avoidance

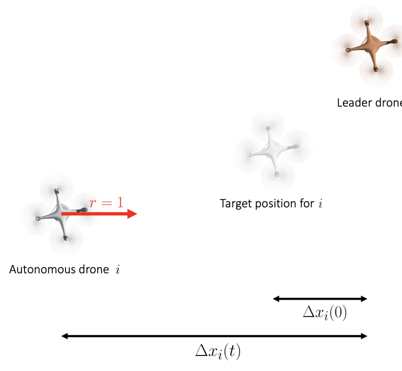
In addition to the above protocol, we can equipped drone with sensors like LiDAR or ultrasonic sensors to detect and avoid obstacles. Autonomous drones can also continuously communicate to each other to share their relative positions to each other. This collision avoidance system was not implemented in our Python Code.

## 5 Horizontal and vertical distances

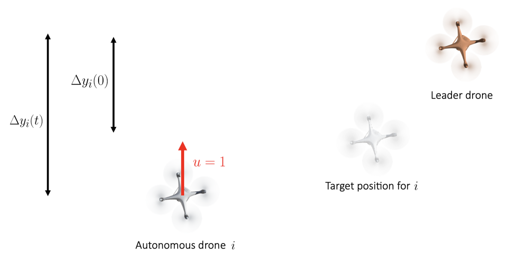
Now that we established the theory of how the drones move, it is important to determine appropriate numerical values for horizontal and vertical distances (these distance are defined in 1). These distances characterize the proximity of the drones during flight. They must be sufficiently large to avoid collisions but small enough to ensure a good camera coverage. Indeed, we want slight overlaps in the drones' captured areas to avoid gaps in the imagery. The determination of these distances is influenced by :

— internal factors such the maximum speed of the drones (fast drones would need more space between each other) and the signal periodicity of the leader drone $\delta$ (if the leader drone communicates frequently with the others, they would be more reactive and thus horizontal and vertical distances can be reduced)

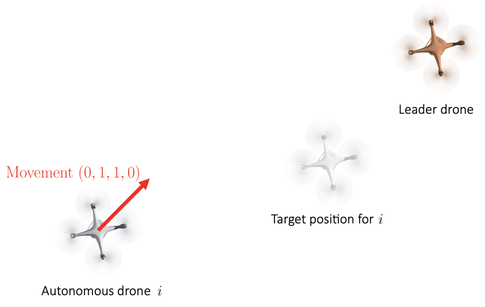
1. "Stop moving" does not necessarily imply that the drone becomes completely stationary. With inertia, the drone may keep some momentum. Hence, "stop moving" can be understood as easing up on the accelerator in a car



(a) Checking horizontal movement



(b) Checking vertical movement



(c) Final next movement signal

FIGURE 2 – Example of application of the movement rule

— external factors such as the wind conditions

## 6 Leader drone path for orthomosaics

Having a rectangular map to cover, a good path strategy is to :

1. Begin at one corner. Let's say the top left hand corner
2. Reach the bottom edge of the map
3. Wait a little bit to allow the autonomous drones to catch up
4. Move to the right of a distance of 5 to 6 times the horizontal distance
5. Go up until reaching the top edge of the map and then shift right again
6. Repeat this process from step 2

By doing only either vertical or horizontal movements (avoiding diagonal ones), we limits the risk of collisions as the drones are positioned diagonally relatively to one another.

## 7 Implementation (in-depth stage for the hands-on option)

We developed a simulation that incorporates all the physics behind (including inertia). This simulation is run using on `Pygame` and shows a top-down view of the drone fleet.

**Here is the github repo :** https://github.com/soniquentin/CCCS660_McGill_Assessment5

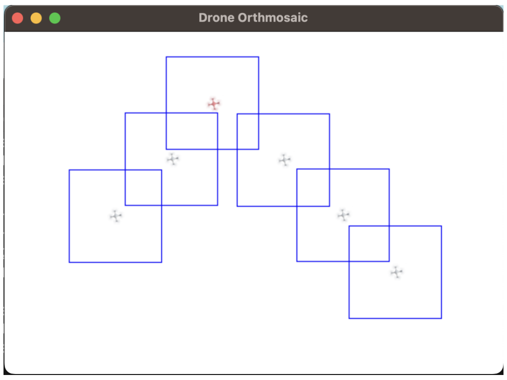


FIGURE 3 – Simulation window (the blue boxes are the video capture areas of each drone)

## 7.1 Quick code structure

The simulation's code is organized into three main files :
- `main.py` : the main python file that runs the simulation
- `drone.py` : contains the class `Drone` and Communication_wave (the signals emitted by the leader drone)
- `params.json` : a configuration file that stores all the simulation parameters. The user can play around with those to experiment with different settings.

## 7.2 Run commands

First install the dependencies :

```
# pip install - r requirements.txt
```

To run a simulation :

```
# python main.py
```

Adjust the simulation settings by editing `params.json`.

## 7.3 Parameters of `params.json`

Here is an explanation of all parameters :
- `width` : width of the simulation window
- `height` : height of the simulation window
- `capture_width` : width of the capture area of each drone
- `capture_height` : height of the capture area of each drone
- `show_capture` : boolean to display the capture areas
- `mass` : mass of a drone (influenced its inertia)
- `speed_max` : maximum speed of the drone
- `horizontal_distance` : horizontal distance between drones
- `vertical_distance` : vertical distance between drones
- `security_zone` : distance to avoid the oscillation problem
- `wave_frequency` : periodicity of the communication waves sent by the leader drone
- `wave_speed` : speed of the communication waves
- `show_waves` : boolean to display the communication waves
- `wind_speed` : speed of the wind affecting the drones
- `wind_angle` : angle of the wind direction in degrees

As one can see, we added wind !

## Références

[1] https://en.wikipedia.org/wiki/Doppler_effect