Open University of Cyprus

Faculty of Applied Sciences

Postgraduate (Master's) Programme of Study

Cognitive Systems

Postgraduate (Master's) Dissertation

Cognitive Swarm of Drones Search and Rescue System

Guillaume Voirin

Supervisor

Loizos Michael

June 2020

Open University of Cyprus

Faculty of Applied Sciences

Postgraduate (Master's) Programme of Study

Cognitive Systems

Postgraduate (Master's) Dissertation

Cognitive Swarm of Drones Search and Rescue System

Guillaume Voirin

Supervisor

Loizos Michael

The present Postgraduate (Master's) Dissertation was submitted in partial fulfillment of the requirements for the postgraduate degree

in Cognitive Systems

Faculty of Applied Sciences

of the Open University of Cyprus.

June 2020

Summary

In an attempt to solve search-and-rescue problematics such as rescue time and difficulty in accessing certain search areas, a cognitive swarm of drones system is proposed, using artificial intelligence techniques interacting with cognitive components.

The system's various elements (drones' cognition, pathfinding, policies, but also humans-swarm interactions) are elaborated, implemented and evaluated using a simulator custom-built for this dissertation.

Evaluation outcomes show that cognitive functions can be beneficial to non-cognitive drone components, and vice versa. Possible improvements are discussed.

Acknowledgments

Thanks to my girlfriend,

to my family and friends for their support in my endeavors,

and to my supervisor Loizos Michael for his advice.

In addition, I would also like to acknowledge support from the European Regional Development Fund and the Republic of Cyprus through the Research and Innovation Foundation with grant number INTEGRATED/0918/0032.

Contents

Chapter 1

Introduction

In the last decade, numerous disasters whether natural (hurricanes, tsunamis, earthquakes, floods, wildfires, volcanic eruptions...) or technological (nuclear power plant explosions, bridge collapses...) have re-emphasized the critical importance of Search and Rescue (SAR) operations.

SAR, according to the US coast guard, can be formally defined as "the use of available resources to assist persons or property in potential or actual distress".

The SAR task can be split into four distinct subtasks — at least:

- Navigate to given location while avoiding obstacles (*pathfinding* actions)
- Explore an area as a team (coordinated *exploration*)
- Identify entities to rescue (synchronized *identification*)
- Manage resources (ensure *autonomy*: eat/drink/rest, charge batteries, refuel, etc.)

Of all the influencing factors, time is arguably the most important in deciding whether SAR operations turn out successful or not (Adams et al. 2007). In fact, according to the aforementioned study, the probability of finding survivors decreases exponentially as time passes, and drops to nearly 0% after only 24 hours. More generally, it seems safe to say that the first 72 hours of SAR operations are critical (Erdelj et al. 2017; Communications 2012).

A number of factors negatively influence rescue timings: they include, amongst others, terrain — or access difficulties in urban areas (Statheropoulos et al. 2015), rescuers' safety, unfavorable climatic conditions, and night operations.

Today, there are two main rescuer profiles on the SAR operations field: humans and robots. Humans are clever: they can use any clue or information to estimate likelihoods of finding survivors in various areas. They also naturally express empathy towards survivors: this helps in guessing their intentions.

They can also try to communicate with survivors. They can carry life-saving equipment and provide emergency medical support, and use tools such as search dogs to benefit from exceptional natural olfactive capabilities. They exhibit inefficiencies in terms of search range though: humans are slow, subject to fatigue or injury, and cannot access certain areas. Robotics, on the other side, are very good in terms of search range as they are able to operate 24 hours a day, and have been used for SAR operations in the last few decades. They are, however, not very clever yet, express no empathy, and cannot communicate directly with survivors (although they can be used to relay communications).

Since the beginning of this century (Murphy 2014), a subset of robotics has been introduced on the SAR operations field in part due to their exceptional search range: unmanned aerial vehicles (UAVs). UAVs (drones) are fast, relatively compact, agile, dark vision-enabled (using LiDAR[1] or thermographic cameras), and can fly into the most remote of areas. They seem like an ideal tool for performing exploration operations, and several UAV-centric non-cognitive SAR solutions can be found in the literature (Erdelj et al. 2017; Waharte and Trigoni 2010; Scherer et al. 2015; Remy et al. 2013). They however exhibit limited autonomy (recharging is often required) — although this is bound to improve; and can hardly carry equipment due to flying-induced weight restrictions.

Current SAR drone swarm approaches, however, do not come without drawbacks. First, a survey report (Chung et al. 2018) suggests that the missing component of modern automated SAR systems is a measure of learning to provide better autonomy and better flexibility. Furthermore, a meta analysis review (Hocraffer and Nam 2017) outlines several human factors challenges, including high cognitive demands for the operator and non-intuitive behavior. This review also states that these challenges could be mitigated by raising the swarm's level of autonomy to reduce operators' cognitive workload and by way of fact improve their situation awareness.

A possibly overlooked aspect of the SAR challenges also lies in the potential absence of well-defined search perimeters (both in area and in nature) at the beginning of operations. How could the uncertain but still critical information-gathering time immediately following a disaster be used in a proactive and useful way, instead of delaying UAV swarm deployment until more information is gained? How could drones, instead of being pre-programmed beforehand to search for specific targets, be enabled to explore and learn from their environment without a definite goal, and ask queries to a human operator — which is, by all means, the definitive expert in determining what is normal from what is not — before triggering rescue interventions?

[1] Light Detection and Ranging

In an aim to overcome these challenges, and **combine the best of both human and automated solutions**, we hereby propose a **cognitive swarm of drones SAR system**.

The main objective is to benefit from search range and reach of UAVs, while integrating a measure of cognition ("cleverness"), but still use human expertise when it is by far the best option.

Our system should be able to learn both from the environment (to enhance functionalities such as *pathfinding* and *exploration*), and from a human operator (to reduce required interactions and lower operator cognitive demands for *identification*). It should display a good level of *autonomy*, by for example managing its charging needs by itself. It should be governed by algorithms that are simple enough — and eventually inspired from natural processes we are familiar with — to be grasped without difficulty, to avoid inexplicable or uninterpretable behaviors and save on resource-hungry onboard computing hardware (and thus on recharging frequency: drones technically cease SAR operations while recharging).

1.1 Challenges

Designing a cognitive swarm of drones SAR system raises a number of problems to be solved.

First, and to adopt a holistic point of view, it seems useful to determine the actual components involved in the proposed system and its operations, whether endogenous (internal to the system, i.e. certain types of drones obviously, but this could also include battery charging stations, radio transmission vehicles, etc.) or exogenous (external to the system, i.e. its environment).

The swarm of UAVs has to be able to *learn* from its environment if it is to be *cognitive*. Cognition, as the definition goes, is the "process of acquiring knowledge and understanding through thought, experience, and the senses" (*Cognition | Definition of Cognition by Lexico* 2020). This can result from individual efforts (each drone acquiring and using its own knowledge to enhance its functionalities), collective efforts (segments of each drone's knowledge being shared and used by all), or both.

However, accumulation of knowledge is pointless without proper use. Therefore, it has to be determined *which* UAV systems can make use of the acquired knowledge and in *what ways*. Could pathfinding be enhanced by cognition? Or inter-drone organization? Operations speed?

Speaking of which: pathfinding is, in fact, the next obvious challenge. SAR operations include a great measure of exploration, and as drones will evolve in

partially known environments, they must be able to handle the following tasks:

- avoid unknown obstacles on their way to a designated (possibly self-designated) target position
- avoid eventual pre-defined, known obstacles (for example, restricted military areas, or more broadly, any areas system operators wish to avoid)

In addition, UAVs generally need to recharge often due to their reduced onboard battery size: weight, of all limiting factors, is arguably the most important for flying devices. Techniques to decide on when to cease SAR operations and return to a charging area must be devised. And, in a more general way, so do behavioral policies which ensure immediate adequacy of UAV behavior within the SAR operations environment.

Also, the human operator - UAV swarm relationship must be worked on, as it is arguably a weaker point in UAV swarm systems.

Finally, to validate chosen solutions, a realistic software simulation should be set up, in order to test and evaluate various components of the system and the system as a whole.

Chapter 2

Related works

Works related to the use of groups of UAVs for general SAR operations are not plethora. This is probably due to at least two reasons: first, as the technology is quite recent, systems are still "handcrafted"; and second, as field testing has thankfully been limited overall (Murphy 2014), it is difficult to gather definitive conclusions on the benefits of using or not certain methods, or their advantages and drawbacks. However, separate works on pathfinding, swarm intelligence, cognitive systems, robotics, and exploration policies are numerous.

2.1 General works

A holistic review (Erdelj et al. 2017) presents various ways UAVs have been used in the context of disaster management. These obviously include search missions, but also monitoring for early warning signs, disaster information gathering, situational awareness and logistics / evacuation support, standalone communication systems, or damage assessment.

Another work (Waharte and Trigoni 2010) evaluates potential UAV search strategies in the SAR context. Three are tested (via a software simulation) against time of finding survivors: the greedy heuristics approach in which each UAV explores the highest confidence location nearby, the potential-based approach in which goals are associated with attractive potentials and obstacles with repulsive potentials, and finally the partially observable Markov decision process approach. Each strategy can lead to good results, however best timings come with exploiting information obtained during search operations to the best extent possible.

A few practical studies (studies which include field testing with actual UAVs) can also be found. The first combines centralized and distributed decision-making methods (Scherer et al. 2015). Another (Remy et al. 2013) evaluates

UAV swarm search strategies (independent or in squadron) with communication constraints. One last presents an UAV multi-agent autonomous exploration solution that ensures real-time adaptability (Sampedro et al. 2016).

Finally, UAV area meshing (enabling UAVs to self-organize in order to optimally cover an area while maintaining communication links), with applications in SAR scenarios, is also explored by a more recent study (Ruetten et al. 2020).

Further works of interest — not necessarily linked to SAR — include a general survey about aerial swarm robotics (Chung et al. 2018), in which various algorithms and state-of-the-art methods for swarm communication, tasks allocation, trajectories planning, and flight coordination are exposed. Flocking algorithms — in particular boids-type algorithms (Reynolds 1987), amongst others, are presented. The report overviews control mechanisms, including the leader-follower paradigm with human-UAV leader(s) interactions and herding strategies. The multiple UAV, multiple moving targets problem that is to be worked on in the present dissertation is also accounted for. Although very difficult to solve in practice, a few approaches including PHD[1] filters (Reid 1979) are cited. The power management issue is also introduced, as well as obstacle avoidance methods (including speed adjustments, sequencing, and potential fields). Overall, the one important area of study that the report identifies as crucial for the future of swarm developments is the introduction of *learning* and decision-making architectures that can provide UAV swarms with higher levels of autonomy and flexibility.

Regarding multi-robot systems, an interesting survey (Senanayake et al. 2016) presents an overview of numerous swarm intelligence algorithms, namely particle swarm optimization (PSO), bees algorithm, artificial bee colony optimization, the classic ant colony optimization algorithm, bacterial foraging optimization, glowworm swarm optimization, the firefly algorithm, and the biased random walk algorithm. These will be inspirational for certain algorithms designed in this Master's dissertation.

Another interesting work — related to the previous — deals with how natural entities move and sense their environment collectively (Berdahl et al. 2013). By taking an example of fishes searching for darker areas, this work shows that a simple algorithm using only local information can describe a collective behavior. This could help in justifying simplified, locality based algorithms in the design of collective systems, as nature and evolution have taken that path more often then not, with great success (bird and fish flocks, bees, ants, etc.). This trend is reinforced by another research work (Lim and Rus 2012) which shows that a global traffic congestion problem can be solved using local information and local processing only.

[1] Probability Hypothesis Density

In the field of robotics and cognition, another work (Levesque and Lakemeyer 2008) outlines the various challenges faced by cognitive system designers in robotics and tries to reconcile traditional, less cognitive robotics with other areas of AI such as planning and agent-oriented programming. Sensing, knowledge representation (or perception), reasoning and planning are introduced with various algorithmic solutions.

Another interesting approach attempts to combine biologically-inspired swarm behaviors with functionalities made possible by technological progress (Alfeo, Cimino, and Vaglini 2019) by offloading some processing from the swarm. In the work's presented search scenarios, technology cooperates well and even enhances nature-inspired search algorithms. Methods reviewed include random walk, human adaptation and an externally computed differential evolutionary algorithm using aggregated swarm parameters to search for the most efficient coordination setting. This work is inspirational for the collective processing components detailed in our proposed system.

All in all, however, the approaches taken by UAV-centric works include few learning and cognitive elements whereas the Chung et al. survey (Chung et al. 2018) has clearly identified a lack in that area: here, we intend to design an SAR swarm of drones system that not only integrates algorithms derived from non-cognitive robotics, but also benefits from learning components in an attempt to better use those algorithms.

2.2 Domain-specific works

To create a *cognitive* swarm of UAVs, where else to look for cognition examples than in our own abilities? Works on human cognitive psychology, including memory and attentional systems (Baddeley and Hitch 1974; Atkinson and Shiffrin 1968; Kahneman 1973; Treisman 1964) provide starting points. It is generally accepted that human memory consists in at least two storage means: short-term and long-term (Atkinson and Shiffrin 1968; Baddeley and Hitch 1974). Furthermore, the manner in which sensory inputs are analyzed is also important for our future cognitive implementation (Treisman 1964; J. A. Deutsch and D. Deutsch 1963). A general and reusable structure of cognitive systems is also usefully presented in a study on human argumentation (Kakas and Michael 2016).

Obviously, **pathfinding** is of significant interest for UAVs to move and explore an SAR operations area. Robotics reactive pathfinding has been a subject of interest for decades — by *reactive*, we mean the task of avoiding obstacles for which a robot has no prior knowledge of. To solve this obstacle avoidance problem, two main approaches have been developed: the first consists in choosing

a direction that seems mostly obstacle-free (the vector field histogram or VFH methods) , and the second in creating a "repelling forces" field from observed obstacles.

The VFH family of algorithms (J. Borenstein and Koren 1991; I. Ulrich and J. Borenstein 1998; Iwan Ulrich and Johann Borenstein 2000) is closely linked to the development of multi-angular obstacle detection systems such as sonar or LiDAR[2] systems (*Lidar* 2020). These latter systems use a rotating laser to calculate a distance from a laser source to an eventual obstacle, by determining how much time it takes for the laser's reflection (if any) to return to its source. When represented on a polar graph, using regular angle intervals, these timing returns form a *histogram* which in effect is a map of close obstacles' proximity. The VFH algorithms utilize this histogram to determine a best direction using various methods (early approaches select the closest obstacle-free direction, while later works account for robots' maneuverability as well).

Repelling force fields algorithms (Khatib 1986; Johann Borenstein and Koren 1988), on the other hand, generally aim at "pushing away" robots from obstacles, while "pulling" them towards their given target T. They can be adapted to high-speed robots, as they include a speed management component.

Planned pathfinding is not to be forgotten: sometimes, certain areas of the map are known in advance. For this task, well-known graph search algorithms exist (a known map can be divided into nodes and connections between each form a graph) such as $A*$ (Hart, Nilsson, and Raphael 1968) and others (Daniel et al. 2010; Nash, Koenig, and Likhachev 2009). These algorithms even can, under certain conditions, find a guaranteed optimal graph path using provided heuristics.

Exploration is another important component of our SAR system. If one considers the SAR operations area to be subdivided in a grid, with each grid cell connected to its neighbors, then exploration consists in visiting as much cells as possible *efficiently* in the graph uniting all cells. As there is initially no knowledge of the environment, exploration will carry a sense of stochasticity at first, and will thus be derived from a pure "random walk" process. Random walks consist in blindly and randomly exploring geographic locations or graphs and more efficient exploration derivatives have been studied to some extent (Pelánek et al. 2005).

Identification of targets to rescue also needs to be considered. This will necessitate the SAR swarm of UAVs system to *learn* from operator indications, and to learn *quickly* as the amount of samples to learn from could be scarce. Decision trees are well tailored for such a use case, and numerous decision-tree creation algorithms exist (Quinlan 1986; Breiman 1984; Quinlan 1993; Kass

[2]Light Detection And Ranging

1980).

Finally, resource **autonomy** problems are necessarily related to drones physics (the heavier a drone, the more energy it requires to fly); and thus, by way of fact, to physics in general. For this, Newton's general theory of physics, although dated, is still a perfectly acceptable approximation of real world physics (*Newton's Laws of Motion* 2020).

Chapter 3

Methodology

First, it is necessary to define what the SAR field is made of, before a system tailored for operations within can be discussed. At first sight, it seems that the amount of possible cases is considerable, as geographic features (plains, seas, mountains, forests, cities, etc.) can vary, and post-disaster areas can be chaotic and in a totally abnormal state (flooded, covered with ashes, debris...).

However, every SAR situation has a set of common features:

- an imaginary **boundary** which defines the area where rescue operations must take place
- a (possibly great) number of static **obstacles**, that can take the form of terrain features (hills, mountains, cliffs, debris, etc.) or man-made features (buildings, poles, etc.)
- a number of areas (**restricted areas**) that should be avoided by exploring rescue devices (above an erupting volcano, near a waterfall, inside protected military areas, etc.)
- living beings and other entities (**unknowns**) whose number and nature are yet to be known, might move and could necessitate a rescue intervention

Unknowns might be rescuable if their features match information provided by human operator(s) during SAR operations.

3.1 Proposed system overview

Our proposed system consists in a swarm of cognitive quadcopter UAVs with a set of landing pads (at least one landing pad per drone) and an operators' station.

Quadcopters seem like the better choice: although they have less range than fixed wing UAVs, their agility and reduced size can be an invaluable asset. They can operate, for example, in cramped urban environments, inside buildings, underground, etc.

Actual exploration and SAR operations are thus performed by quadcopter UAVs while landing pads enable deployment and recharging, as well as radio transmissions between drones and the human operators' station. For practical reasons, operators might not necessarily be present on the SAR field of operations. Vehicles (like trucks, or ships for maritime SAR operations) could carry landing pads, and give a good balance of deployability speed, mobility and versatility to the system.

3.2 Cognitive drones

Why, in the first place, equip drones with **cognitive** capabilities?

Simply put: the general idea is to add a learning and abstraction layer after environmental sensing, and use this abstraction layer to enhance overall functionalities of UAVs, as depicted in figure 1 (where policies can consist in procedural actions, pathfinding, etc.).

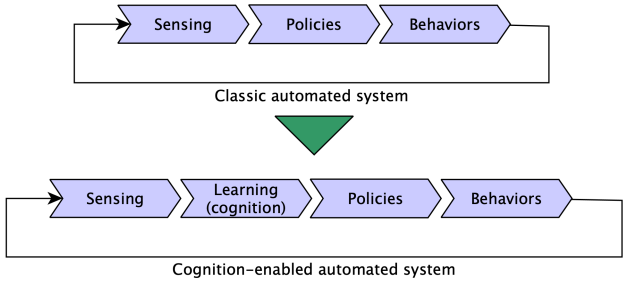


Figure 1: Classic vs. cognition-enabled automated systems

3.2.1 Sensing

As the schema shows, everything starts with *sensing*. Drones are thus equipped with the following:

- a sense of **location** (via a GPS[1]-like system)
- a sense of **vision** (via a camera and image analysis system)
- an **obstacle proximity** sense (via a LiDAR system, which operates like a bat's sonar but with lasers)

[1] Global Positioning System

- a **radio reception** sense (a radio system)

Additionally, a sense of hearing using a stereo microphone can be added for further bias confirmation / entity recognition, but is not mandatory. A few things can be elaborated regarding camera and LiDAR systems:

Camera and image analysis system

The image analysis system is capable of distinguishing essential features (elements) of the environment. CNN[2]-based vision systems, for example, display good results (Krizhevsky, Sutskever, and Hinton 2012; Szegedy et al. 2015; He et al. 2016).

LiDAR

LiDAR[3] is a system using light in the form of a pulsed laser to measure distances to obstacles (*Lidar* 2020) as pictured in figure 2 where a laser beam is rotated from left to right, and the position of terrain features computed using timing information (the quicker a laser light comes back to the emitting source, the closer an obstacle is).

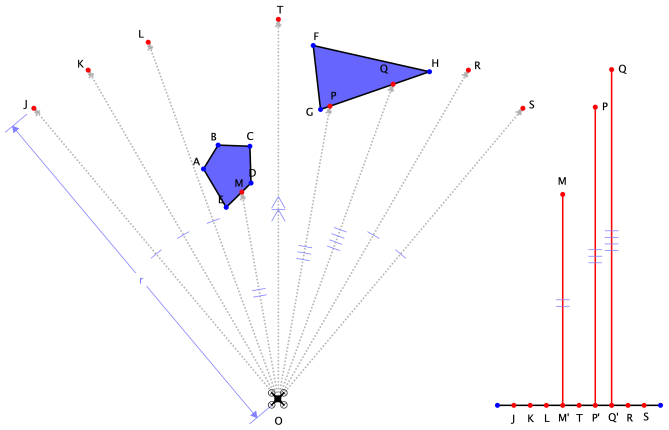


Figure 2: A 90 degree LiDAR scan and associated obstacle-scape result

Scans OJ, OK, OL... are performed in sequence, by a laser rotating from left to right. Range r is the LiDAR operating range beyond which obstacle detection returns are inaccurate and ignored. JKL, T and RS represent gaps, whereas MM', PP' and QQ' are obstacle returns.

[2]Convolutional neural networks
[3]Light Detection and Ranging

3.2.2 Cognition

S*ensing* is not *perceiving*. For example, the amount of visual information that reaches our retinas is — simply put — colossal, and yet, our brain manages to extract important features from the flowing magma of data its vision senses provide: it perceives the sky, the sun, a bird. But it has discarded the small, unimportant stain on the window.

This simplified example outlines the filtering process that is taking place between our senses' raw capture of information and our memory's ability of presenting this data in a conscious or unconscious form, which is none other than the perception and attention mechanism. Perception and attention are closely interlinked: attention can modulate perception (for example, when focusing on the road, accident likelihood decreases (Kahneman, Ben-Ishai, and Lotan 1973); or when focusing on a basketball, we don't perceive the gorilla (Simons and Chabris 1999)), and perception can direct attention (for example, an alarm sound will be perceived without being attended to but perception will immediately shift our attentional focus on it).

Numerous human cognitive psychology research works based on visual (Wolford and Morrison 1980; FitzGerald and Donald E. Broadbent 1985) and auditory (Cherry 1953; D. E. Broadbent 1958; Treisman 1964; J. A. Deutsch and D. Deutsch 1963) experiments have been conducted to evaluate the amount of brain processing on unattended signals, and although theories may differ, it has been globally shown that there is, at least, a certain quantity of brain processing on unattended inputs, but that this processing is more intense for attended inputs (Coch, Sanders, and Neville 2005). In other words, perception of at least some unattended signals (if not all) is actioned by our mental processes.

Here, we will take the bias of having drones **perceive all information**: everything drones' senses capture will be perceived, but only specific elements of what is perceived will be attended to or stored. It is noteworthy to point out that attention, in our system, is not used as a sensory input filter like in human cognition, but rather as a means to emphasize or prioritize certain elements in working memory.

Overall, drones benefit from two distinct cognition abilities: an individual cognition ability (sourced from the drone's own senses, attention/perception, and memory mechanisms) and a collective cognition ability (sourced from every drone, propagating via radio broadcast some of its perceptions to the swarm as a whole). An overview of the swarm of drones' cognition system is schematized in figure 3 and different parts detailed in the forthcoming paragraphs.

Finally, cognition would be relatively purposeless if it wasn't for the process of creating or updating a plan to successfully achieve a set of goals: every drone

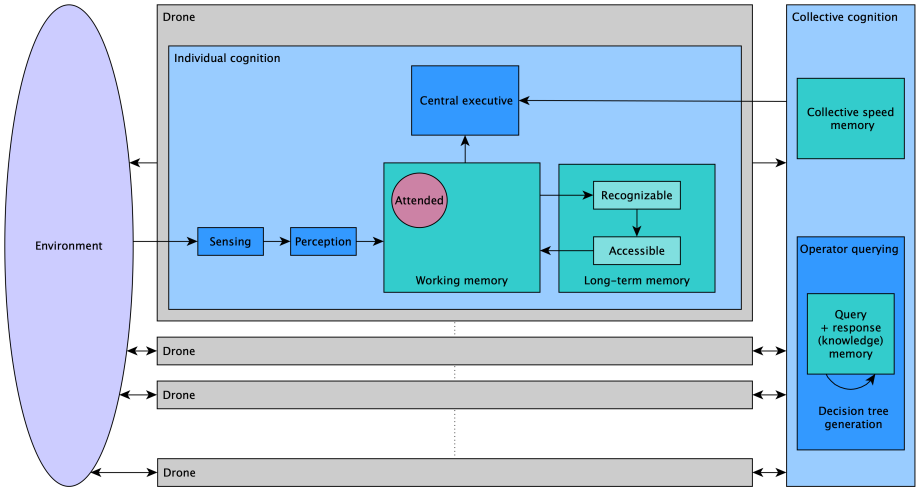


Figure 3: Drone swarm cognition overview

has an abstract set of *intentions* (i.e. goals) and an associated set of policies to reach those goals. This is elaborated in section 3.2.4.

Individual cognition

Perception It is noteworthy to elaborate on an important process of perception, which is the "integration" capability. When we witness a barking dog, we immediately associate sound and image as a single dog entity. However, this is not trivial: how would we perceive a meowing dog? In any case, integration is not really an issue here as perception systems are linked to different sets of functionalities (it is not really important if an obstacle and a living being, for example, are not integrated into a single entity: in the end, the obstacle will be avoided and the living being rescued).

Memory One could argue: drones gather interesting information, let's just store *everything* and work from there! Why bother with a hierarchical memory system?

This is not, however, the philosophy of our approach: nature has taught us that only what really matters suffices in our adaptation to the environment, so why take into account everything? We don't, and we are still the best SAR system known to date.

Furthermore, there is always a cost in not properly tailoring computer systems for their intended use: in the present case, drone energetic autonomy would suffer (and so would operational time) if battery usage were to be increased due to elevated amounts of data storage and associated processing. Also, storage

pre-filtering helps in achieving an immediate notion of *saliency* and this can be life-saving (imagine if, for example, we stored an alarm sound with numerous other pointless informations, and only processed it an hour later...).

Drones are therefore equipped with a dual store memory system loosely derived from the multi-store model (Atkinson and Shiffrin 1968), or to some extent, Baddeley and Hitch's working memory model (Baddeley and Hitch 1974).

They possess a working memory and a long-term memory, complemented by a separate spatial memory. Distinct elements from working memory and accessible long-term memory constitute what is referred to as the "**mental model**" of the drone. Elements in working and long-term memory are used to store salient environment information, and reuse it even if it is not sensed anymore — or if some senses were to be dysfunctional, just as we can manage to find our way in the dark inside places we know.

There is no sensory memory so to speak, as every sensed world component is immediately processed by perception and stored in working memory, and removed from it soon after perception stops.

Working memory Perceived elements are added to working memory immediately following perception, and are removed after not being perceived for a lapse of time δ_{wm}. Perception time is not irrelevant to human cognition: we are inclined to learn things we see often, although this is far from being an absolute truth as learning is influenced by numerous other factors.

Long-term memory Long-term memory consists of two distinct storages: a "recognizable" storage and an "accessible" storage.

Everyone has experienced recognizing an old image (old photo, etc.) although it had been completely "forgotten". Although memorizing and forgetting processes are an active subject of study and give lieu to various theories, it is undeniable that retrieval cues (such as the old image itself in the above example) play an important role in accessing long-term memory (Goodwin et al. 1969; Jonker, Seli, and MacLeod 2012): in other words, we might actually hold much more knowledge in long-term memory than we think — "recognizable" if given proper clues but not necessarily "accessible" at will. Drones' long-term memory structure is based on this paradigm.

The "recognizable" storage receives all elements that enter working memory. In other words, once an element has been perceived by the drone and has thus entered its working memory, it is also stored in the "recognizable" area of long-term memory. After perception, an element is permanently recognizable by the drone, but is not yet accessible at will.

The "accessible" storage represents elements in long-term memory that are

immediately accessible and thus part of the drone's "mental model". They represent entities of the environment that the drone has learned as they are often perceived, for example a major obstacle in the center of the map that is being avoided continuously. To become a part of the "accessible" storage, an element has to reside in working memory for a minimum time δ_{acc} such that $\delta_{acc} \gg \delta_{wm}$, to ensure that elements stored in accessible long-term memory are significant to each drone. Significance, in the present context, is linked to the amount of time an entity resides in working memory.

Spatial memory Much like one of the hippocampus's recognized functionalities (O'Keefe and Dostrovsky 1971), drones are equipped with a location or spatial memory. This memory is subject to decay as only a limited amount of locations are stored. When spatial memory is full and a new location is visited, the oldest visited location is forgotten.

Attention

Attentional systems and their filtering of information are usually beneficial for at least two reasons:

- without any pre-conceived knowledge, they offer root mechanisms for learning, by bringing salient information to the front, to allow for further analysis or storage
- they reduce required parallel analysis capabilities (i.e. information processing resources) by shunting a good amount of non-significant (or perceived as so) information

Both these advantages are possibly utilized by the human brain to better manage its resources; and although the *why* of attention is yet to be fully known, it has survived the test of Darwinian evolution. Its obvious drawback lies in the discarding of considerable amounts of information which might contain critical but unprocessed clues (magicians and pickpockets alike make a living out of attentional drawbacks).

In practice, when an element is added to or removed from working memory, it is tagged as attended and stays so for a minimum lapse of time δ_a (if an element is removed from working memory, it is still stored temporarily as long as it is attended to but not considered part of working memory). However, if an attended element sees its status changed in working memory, this timing is reset and the element stays attended for a new lapse of time of at least δ_a seconds. This ensures that all attended elements are *salient*: their presence and absence changes dynamically over time and these changes create new information which require elucidation.

Summary

The above processes run in parallel and are assembled in algorithm 1, with D representing a drone.

Algorithm 1 Individual cognition mechanism

```
function IndividualCognition(D)
    do in parallel
        E ← Perception(D)
        for e ∈ E do
            if e ∉ WorkingMemory(D) then
                WorkingMemory(D) ← add(e)
            end if
            if e ∉ RecognizableLTM(δ) then
                RecognizableLTM(D) ← add(e)
            end if
        end for
        for w ∈ WorkingMemory(D) do
            if Age(w) > δ_wm then
                WorkingMemory(D) ← Remove(w)
            end if
        end for
        for c ∈ Changes(WorkingMemory(D)) do
            if c ∈ Attended(D) then
                Reset(Age(c))
            else
                Attend(c)
            end if
        end for
        for a ∈ Attended(D) do
            if Age(a) > δ_a then
                UnAttend(c)
            end if
        end for
        for r ∈ RecognizableLTM(D) do
            if time(r, WorkingMemory(D)) > δ_ltm then
                AccessibleLTM(D) ← add(r)
            end if
        end for
        SpatialMemory(D) ← add(Location(D))
        if |SpatialMemory(D)| > ζ_sm then
            SpatialMemory(D) ← remove(oldest)
        end if
    end do
end function
```

Collective cognition

As a reminder, "collective cognition" refers to a set of cognitive processes that take place onboard *each* drone but that use perceptual, radio-transmitted data from *all* drones of the swarm.

Collective speed memory As drones' efficiency in moving in their environment is highly desirable — be it only for battery savings, we introduce in the forthcoming pathfinding section a collective (swarm intelligence) speed-based attraction algorithm, to entice drones to use the "faster lanes" (obstacle-free lanes) of the map.

This algorithm is supported by the knowledge of collective average speed informations throughout the map. Therefore, each drone broadcasts its current

speed when visiting a location, to inform all others. This information is then stored by each drone in a memory called the "collective speed memory".

A location-based diffusion algorithm is continuously applied to this storage space: just as a coloring substance slowly propagates and changes the color of its container water, average speed values modify and "color" the values of their neighbor locations, while respecting the natural diffusion principle i.e. every value increase at a location must be compensated by a combined identical decrease at some other(s) and reciprocally. This diffusion must be sufficiently slow to allow a certain persistent "memory" of units passage to remain, but not too slow to enable the creation of a progressive speed average gradient.

Our proposed diffusion algorithm is stochastic: on every iteration it chooses a random location λ on the map to update its and its neighbors' values. Algorithm 2 lists the pseudocode logic, with $\chi(l)$ the neighbors list function and $\nu(l)$ the average speed value at location l.

Algorithm 2 Collective speed diffusion algorithm

```
function DIFFUSE(λ)
    d ← ην(l)
    n ← |χ(λ)|
    ε ← d/n
    ν(λ) ← ν(λ) − d
    for l ∈ χ(λ) do
        ν(l) ← ν(l) + ε
    end for
end function
```

Where η is a small positive value ($\eta \ll 1$) influencing diffusion speed.

Stochasticity reduces artifact effects and produces better overall results at the expense of slight imprecisions as some locations are temporarily visited more than others, but with time, statistical smoothing preserves overall accuracy. It also provides for a significant advantage of our presented algorithm which is a parallelization ability.

Operator querying The swarm of cognitive UAVs is able to query and learn from human operator responses. Obviously, if every drone were to ask its own queries directly, the human operator(s) would soon become overloaded, hence we introduce here a collective query system, that enables issuance of queries *if* they are novel *and* current accumulated knowledge is not enough for drones' machine learning processes to decide on a probable response (that is, barring scarce stochastic re-issuances as described later).

Environmental observations pre-empting queries are gathered by the vision system and its analysis software using **attention** (only attended-to elements are queryable, in order to limit queries to salient elements), and must be formalized to fuzzy feature vectors to make sure that elements displaying a great number

of similarities are not tagged as distinct (which would clutter the system with redundant queries) because of detail-level differences. Fuzzy feature tags are predefined and could, for example, include attributes of sizes, shapes, attitudes, positions, movements, entity types (humans, animals, objects...), noise types (if drones are equipped with a sense of hearing) etc. — in fact anything deemed appropriate for SAR operations (the broader, the better in this case) with all attributes having an "unknown" feature to account for lack of or perception failure.

Query broadcasting Every time a drone observes an entity in the SAR environment, whose fuzzified feature vector is undecidable using machine learning as to whether or not it needs rescuing, it broadcasts a rescue query via radio. On a random basis, some queries which are associated with clear predictions from machine learning are also allowed to be broadcast again, to confirm operator bias (in other words: that two identical queries get the same answer) and up-to-date validity of the knowledge base (things might change during the course of SAR operations, with incoming field reports).

Queries are received by landing pads and relayed to the operators' station which eliminates duplicates (drones in proximal areas might make similar observations simultaneously).

Knowledge memory Knowledge memory contains a list of fuzzified feature vectors with operator-provided answers (also called *outcomes*). Every time an operator asynchronously replies to a query, landing pads broadcast the feature vector and associated outcome (and, in the case of a negative outcome, an eventual measure of "machine coaching" in the form of an attribute hint), and both are added to every drone's knowledge memory. To use the above attributes example, a fuzzified feature vectors could be: {small, cubic, northwest, static, rock, unknown} and its outcome {no} with an attribute hint {type}. This knowledge base is then used as a training set by machine learning to decide on forthcoming observations.

Machine learning We hereby propose the use of a decision tree-based classification algorithm: the *ID3* (Iterative Dichotomizer 3) algorithm (Quinlan 1986). It has a number of advantages, including:

- it generates understandable prediction rules
- few training examples are required, to create useful decision trees
- with no conflicting outcomes, the resulting tree validates its training set entirely, which is highly desirable in a SAR operations context

Furthermore, the use of operator attribute hints can enable quick generalization, by adding at once an important number of combinatory knowledge statements to knowledge memory. $ID3$'s main drawbacks are *overfitting* as created trees have no size bounds (although this can be mitigated by pruning techniques) and computer intensiveness with bigger datasets, but in the present case this is not a problem as the training set will be reduced in size (otherwise there is an operator overloading problem). The algorithm is used here to predict the classification of *outcomes* depending on the values of other attributes' features.

$ID3$ is based on the concept of *information gain* (equivalent to *entropy loss*) to decide on which node to add next to the tree. Leaf nodes have an entropy of 0.

Entropy $E(S)$ is defined as follows (S is a state i.e. a selection of training set examples from knowledge memory, X is the outcomes attribute, consisting of *yes*, *don't know*, and *no* answers, and $P(x)$ is the probability of an event x of S) :

$$E(S) \quad = \quad \sum_{x \in X} -P(x) \log_2 P(x)$$

Information gain is computed as such (S_v is a subset of S for which attribute A has value v):

$$IG(S, A) \quad = \quad E(S) - \frac{|S_v|}{|S|} E(S_v)$$

The algorithm starts with an empty tree, and iterates until all leaves have a nil entropy (all training set statements lead to a decision). On every iteration, the attribute A that yields the largest information gain is chosen as the next decision node. The tree is recomputed on a periodical basis onboard each drone.

In practice, when an observation receives a predicted *yes* outcome from the onboard decision tree, the observing drone triggers a rescue intervention request by radio including its current position, in order to inform all SAR operations members (operator(s) and other drones to prevent duplicate requests).

3.2.3 Pathfinding

Exploration, and thus pathfinding, is a principal feature of any SAR system. A common pattern found in automatic guidance systems it to have an autopilot directly interpret obstacle sensing, but here, we direct pathfinding to actually take its inputs from the *mental model* of the drone (which, as a reminder, is

constituted of distinct elements from working memory and accessible long-term memory, issued themselves from sensing and perception).

Just as when we lean on a wall, we do not actually see it but we know it is there as we have previously learned from its presence, drones are able to *see* things they don't currently perceive, such as obstacles behind or obstacles far ahead, because they have already been *exposed* to them and have *learned* from their previous exposures' perceptions.

It is considered here that drones operate in a horizontal plane: designing simple but efficient three-dimensional pathfinding algorithms seemed out of scope for the timeframe of this Master's dissertation, especially when the emphasis is put on cognition *and* pathfinding and not on pathfinding alone. Fixed-altitude operations are far from unrealistic anyways: they are required in many areas such as inside buildings, near airports, above certain military areas etc.; also, altitude changes are often convenient but not necessarily efficient battery-wise (Shakeri et al. 2018), autonomy being one of the major drawbacks of quadcopters today; and finally, it is more challenging for a collision avoidance system to cater for constrained spaces. It is however possible to adapt all the presented algorithms to three-dimensional operations as they have no dimension-bound constraints.

At first sight, pathfinding has at least four components:

1. a **collective drone movement** component (in other words, inter-drone avoidance and movement as a swarm)
2. a **dynamic obstacle avoidance** component (avoidance of unpredictable obstacles)
3. an **efficiency** component
4. a **restricted area avoidance** component (avoidance of known obstacles — stored on each drone in a permanent database).

All components need to be combinable. The efficiency component will here be provided by a **collective speed attraction** algorithm to learn from other drones' speeds at various locations.

Overall, this separates the pathfinding problem in two categories: while the first three components attempt to reach a given target T using *dynamic adaptation* to the environment, the last one is a *planning* problem that can be used to precompute lists of intermediate targets T_i to use by the first three.

In order to achieve *dynamic adaptation*, we introduce here a system of sum of forces (or potential fields), where each sub-component (collective drone movement, dynamic obstacle avoidance, and collective speed attraction) commands a

force that is added to the global pathfinding force $\overrightarrow{F_{pathfinding}}$. This process ensures a *fluid* or *analogue* nature to pathfinding: it is progressive and continuous in nature, and easily explainable by studying individual sub-forces.

Pathfinding *planning* is treated as a path optimization problem, with classic artificial intelligence tools.

Collective movement forces

Drone/drone avoidance is, in a way, a special case of obstacle avoidance, but it is useful to distinguish the two as drones gather much more information from their peers than from a generic obstacle: they can perceive other's exact position, speed, intentions, directions, etc. and it is beneficial to use this information to improve collective drone maneuvering and organization.

Here, we use an iteration of the boids algorithm (*Boids (Flocks, Herds, and Schools: A Distributed Behavioral Model)* 2020; Reynolds 1987) as it is compatible with the "sum of forces" paradigm. This swarm intelligence algorithm applies three different forces to every boid — drone in our case (Hartman and Benes 2006): cohesion, separation, and alignment forces.

Cohesion (flocking) Drones try to stay close to the center of the local flock formed by the drone and its neighbors (denoted C in figure 4). Other drones are considered part of the "local flock" if their distance from the current one are inferior to a predefined ρ_c value.

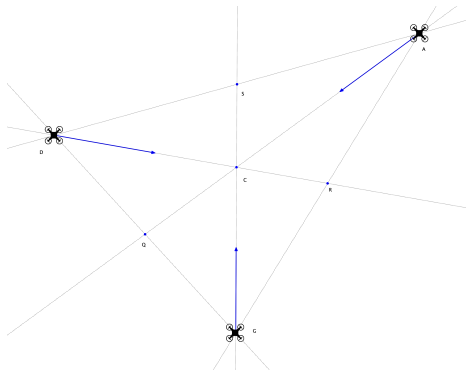


Figure 4: Drone cohesion forces

Position of center C is calculated as follows, if $p_{1..n}$ are the positions of the local flock of n drones:

$$p_C = \frac{1}{n}\sum_{i=1}^{n} p_i$$

Cohesion force is then computed for each drone as follows (with p the position of the drone):

$$\overrightarrow{F_c} = p_C - p$$

This force is only applied to the set of drones having the same *intentions* (i.e. moving to the same point for example).

Separation Drones must stay far enough from their neighbors to avoid potential collisions (as in figure 5). A drone is considered a neighbor if its distance is less than a predefined ρ_s value.

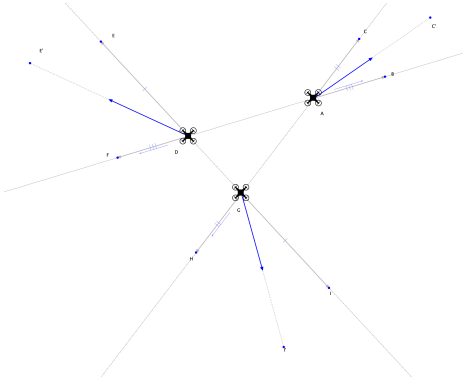


Figure 5: Drone separation forces

With p the position of the current drone and $p_{1..n}$ the positions of its n neighbor drones, we use the following:

$$\overrightarrow{F_s} = \sum_{i=1}^{n} \frac{p \hat{-} p_i}{\Delta(p, p_i)^2}$$

Where $\Delta(p, p_i) = \max(\varepsilon, d(p, p_i) - \omega)$ is the actual distance between two drones (as positions are taken at drones' center of gravity, two drones whose position distances are inferior to a drone width are probably touching each other), with ε a small positive number such that $\varepsilon \ll 1$ and ω the usual drone width.

This formula generates a significant avoidance force when drones are close by, and a negligible one when they are well apart.

Alignment (velocity matching) Drones try to match the direction and speed of their neighbors (a drone is considered a neighbor for alignment if its

distance is less than a predefined ρ_a value) as in figure 6.

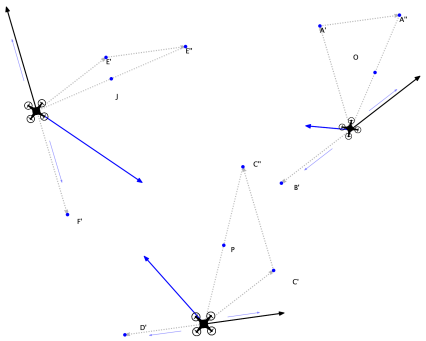


Figure 6: Drone alignment forces

Speed vectors are displayed in black and alignment forces in blue.

With $\overrightarrow{s}$ the current drone's speed vector and $\overrightarrow{s_i}$ other drones' speed vectors, we have:

$$\overrightarrow{F_a} = -\overrightarrow{s} + \frac{1}{n}\sum_{i=1}^{n}\overrightarrow{s_i}$$

Summation Cohesion and separation appear like antagonist forces, but their norms and directions are usually different. Alignment forces encourage drones to move as a *squadron*.

The final collective motion force is computed as follows for each drone:

$$\overrightarrow{F_{cm}} = \alpha_c\overrightarrow{F_c} + \alpha_s\overrightarrow{F_s} + \alpha_a\overrightarrow{F_a}$$

With $\alpha_c, \alpha_s, \alpha_a$ a set of predefined coefficients to fine-tune each sub-force's influence.

Dynamic obstacle avoidance

The problem with dynamic obstacles is that no real planning can be made: they are mostly unknown to the drone, apart from those that have already been encountered and remembered. Thus the system must be *reactive* as opposed to predictive: it must adapt to unforeseen situations in real time, but nothing guarantees that it won't take a sub-optimal path (in the medium to long term that is).

To best cater for this, we introduce here an algorithm derived from the VFH family of algorithms (J. Borenstein and Koren 1991; I. Ulrich and J. Borenstein 1998; Iwan Ulrich and Johann Borenstein 2000). VFH uses information from the returns of a set of ahead-projected rays (which is exactly the information provided by LiDAR systems) to find gaps in the obstacle landscape (directions where there are no obstacles, also called a *polar histogram*). The best gap is then selected in different manners depending on the VFH variant.

Our algorithm is effectively derived from VFH but with one (big) difference: instead of directly using LiDAR-provided data, it **uses cognitive information**: rays are virtually projected via software using the drone's "mental model" information (in effect, this means that data from LiDAR returns *and* memory are used altogether). This can, in fact, make a significant difference in choosing the best direction as drones' lookahead capabilities are much greater than their LiDAR range once they have already acquired a certain "knowledge" of the environment (accessible from their "mental model"). The algorithm is also tailored for high-speed quadcopters (VFH was initially designed to guide relatively slow robots on the ground).

Before dynamic obstacle avoidance computations are made, cognition-provided obstacles are artificially resized via software in the following way: they are inflated with a *safety area* that can be computed in a straightforward manner by "extending" obstacle detection traces in all directions on a horizontal plane as in figure 7, by a length l — usually adjusted to match drone maneuverability (the less maneuverable the drone, the greater the size of the safety area). This is a safety measure to ensure drone pathfinding will not compute trajectories incurring close encounters with obstacles (a gust could easily send an UAV into a wall).

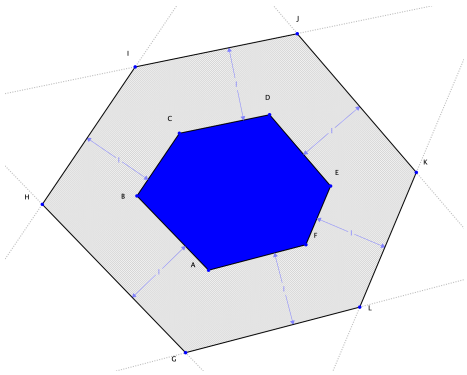


Figure 7: Obstacle $ABCDEF$ in blue and computed safety area $GHIJKL$ in hashed gray

From there, algorithms consider the safety area to "be" the real obstacle and

try to avoid it at all cost.

Static repelling forces Intuitively, it is desirable to stay away from obstacles in general as they might incur abrupt avoidance actions which could outmaneuver fast-moving UAVs, but this must not prevent a drone to reach its target position T if T is near an obstacle.

Therefore, we apply a set of "repelling" forces when a drone is in the vicinity of obstacles:

1. An emergency repelling force that is triggered when drones, despite avoidance algorithms, penetrate an obstacle's safety area
2. A slow-down force introduced when a drone is quickly approaching an obstacle's safety area
3. A tactical placement force that gently pushes drones away from obstacles

Emergency force If, at any time, a drone is perceived inside an obstacle's safety area, with $\overrightarrow{n}$ the closest safety area segment normal as in figure 8 and φ a factoring coefficient (taken quite big as the safety area must be immediately exited), we take:

$$\overrightarrow{F_e} = \varphi\overrightarrow{n}$$

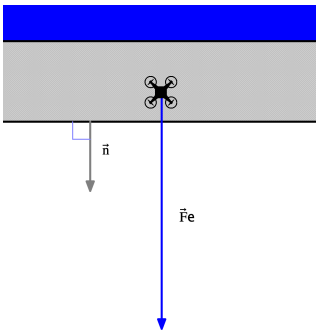


Figure 8: Emergency repelling force

Slow-down force The slow-down force attempts to reduce the incoming velocity of drones quickly approaching an obstacle safety area segment, if this segment's distance is less than target T's (see figure 9). With $\overrightarrow{s}$ the drone's speed and d_i the distance to the intersection of the drone's direction with the safety area segment:

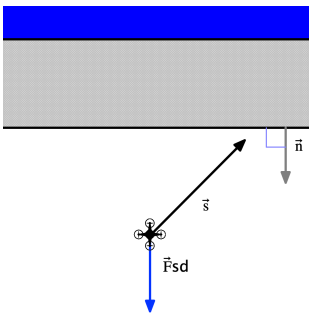


Figure 9: Slow-down repelling force

$$\overrightarrow{F_{sd}} = \frac{-2\overrightarrow{s} \cdot \overrightarrow{n}}{d_i}\overrightarrow{n}$$

Tactical placement force An artificial tactical placement force is created from all cognition-provided obstacle segments and is proportional to their distance to the drone (this is the same concept as the boids separation force above). With p the drone's position, n the number of obstacle segments, and S_i each segment ($d()$ computes a distance from point to segment, as in algorithm 5):

$$\overrightarrow{F_{tp}} = \frac{1}{n}\sum_{i=1}^{n}\frac{\overrightarrow{n_i}}{d(S_i, p)}$$

Summation Overall, the global static repelling force is thus:

$$\overrightarrow{F_{sr}} = \overrightarrow{F_e} + \overrightarrow{F_{sd}} + \overrightarrow{F_{tp}}$$

All sub-forces might be nil depending on conditions (drone not in an obstacle safety area, not approaching an obstacle segment, or not perceiving any obstacle respectively).

Best direction force (VFH-inspired) The best direction force is the principal force of obstacle avoidance, as it attempts to propel drones towards their target T, while "steering" them away from frontal obstacles, by looking for gaps between those obstacles.

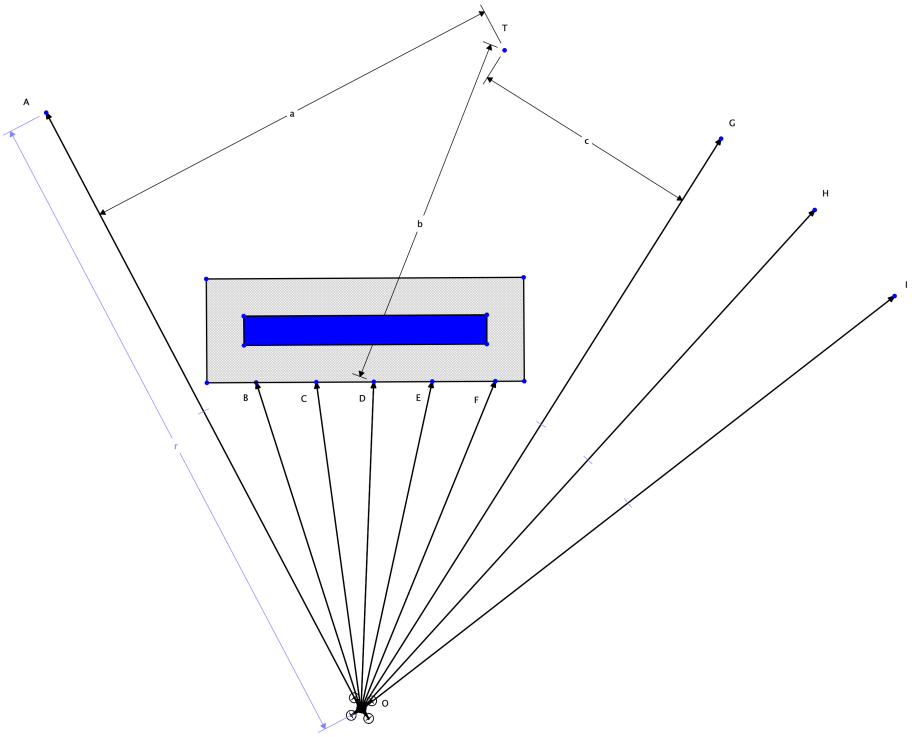


Figure 10: Best direction selection

The straight line towards target T is tested first and immediately used if clear. When this is not the case (such as in figure 10), a choice of direction has to be made. The best direction amongst possible options is chosen according to a segment-to-point distance. In practice, this segment-to-point distance for a segment AB and point P is the distance between P and the point on AB that is closest to P (see algorithm 5).

In figure 10, where O is the drone's position and segments $OA, OB, OC...$ are the software-projected rays (with $r = d(O,T)$), with a blue rectangle obstacle and its hashed computed safety area, we see that the best (lowest) distance possible to T corresponds to distance c or $d(O,G)$, hence OG is considered the best direction available as it can bring the drone closest to its target.

The resulting force, with p_b the position of the best direction segment's extremity (G in the above example, it could simply be T if in line of sight), p and $\overrightarrow{s}$ the considered drone's position and speed vector, and γ_a an anticipation scalar that helps avoid target overshoot, is written:

$$\overrightarrow{F_{bd}} = p_b - (\hat{p + \gamma_a \overrightarrow{s}})$$

Escape maneuver force Our best direction selection algorithm has a drawback: in some corner cases, a drone can get "stuck" near an obstacle; in fact, VFH-family algorithms are known to be sensitive to "local minima" (Leca et al. 2019). This can happen, for example, when drones discover their environment and meet an obstacle that is much wider than the field of view of their LiDAR system (which is the only source of obstacle information available to cognition at this stage, as nothing has been learned yet).

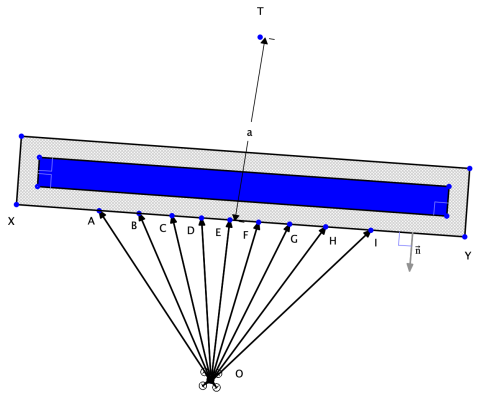


Figure 11: Stuck drone

In this situation (depicted in figure 11), the drone is unable to "see" any gaps in obstacles ahead and thus the selected best direction is OE (as distance a is the shortest possible) which sends the drone straight towards the middle of the obstacle — where it will soon stop without reaching target T due to the influence of obstacle repelling forces.

When this happens, a secondary logic or "escape maneuver" is triggered. The escape maneuver consists in the addition of a force pushing the drone "along the walls" (i.e. the obstacle's safety area in practice): exactly as, when lost in a labyrinth, we follow a wall to find our way out, here walls are followed until the drone either acquires a line of sight with its target T or finds a gap in the drone-target direction. This is similar, in a way, to the "tangential escape" approach (Ferreira et al. 2008; Brandão, Sarcinelli-Filho, and Carelli 2013).

In short, when the drones' average speed gets below a threshold value σ_a, the following force is triggered, with XY the closest obstacle safety area segment (such as in figure 11), $\overrightarrow{v}$ equal to $\overrightarrow{XY}$ or $\overrightarrow{YX}$ (whichever results in the smaller turn), α_{em} a scaling coefficient and $\overrightarrow{n}$ the normal unit vector to XY pointing towards the drone:

$$\overrightarrow{F_{em}} = \alpha_{em}(2\hat{\overrightarrow{v}} - \overrightarrow{n})$$

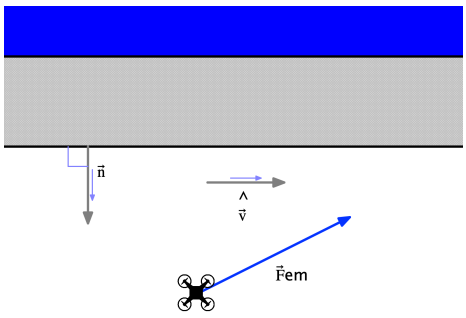


Figure 12: Escape maneuver force

The force also gently pushes the drone towards the wall as in figure 12, to counter static obstacle repelling forces.

Summation Overall, the dynamic obstacle avoidance force is the following, with k a scaling coefficient:

$$\overrightarrow{F_{doa}} = k\overrightarrow{F_{sr}} + (\overrightarrow{F_{bd}} \textbf{ or } \overrightarrow{F_{em}})$$

Collective speed attraction

As we have seen in the cognition section, collective cognition includes a collective speed memory that is used to store average speeds of drones.

The swarm intelligence algorithm using this data and introduced here is based on the ant colonization family of algorithms (Dorigo, Birattari, and Stützle 2006) where, by producing pheromones as they move, ants attract other members of the colony on their path. This, in turn, produces an interesting path-optimization emergent behavior.

If, for example, an ant colony lies in A and a food source in B, as they explore their environment, ants from A will inevitably find various ways to go to B. However, ants using the shortest paths will be able to perform more journeys between A and B than others in a given time, and will hence leave a higher density of pheromone traces of their passage. As pheromone concentrations get increased in the vicinity of shortest paths, other ants will be enticed to use them as well and the process will self-reinforce itself, leading the whole colony to automatically prefer shortest paths from A to B.

However, in the context of SAR operations, drones actually want to *explore* their environment (in other words, display a measure of curiosity) and not go from a point to another using the same path repeatedly (except, possibly, to return to landing pads to recharge): ant-colony algorithms are not applicable as such. We therefore propose a variation based on speed of travel.

Collective speed values are here used to influence drone paths *laterally*: a collective speed attraction force $\overrightarrow{F_{csa}}$ is introduced, perpendicular to a drone's speed vector $\overrightarrow{s}$. As a reminder, collective speed memory is permanently the subject of a diffusion algorithm, whose aim is to create progressive average speed gradients between grid cells of the SAR operations area.

If a drone with position p and speed $\overrightarrow{s}$ is currently inside a grid cell G, its lateral positions p_l, p_r are found by using grid cell diagonal length l_d and $\overrightarrow{s_r}$ a vector obtained by rotating $\overrightarrow{s}$ to the right by a 90° angle in the horizontal plane containing p:

$$\begin{aligned} p_l &= p - l_d \hat{\overrightarrow{s_r}} \\ p_r &= p + l_d \hat{\overrightarrow{s_r}} \end{aligned}$$

From there,

$$G_l = \textbf{gridcell}(p_l)$$

$$G_r \quad = \quad \textbf{gridcell}(p_r)$$

If $\upsilon(G_i)$ is the average speed value at cell G_i, the lateral collective speed attraction force is computed as follows:

$$\overrightarrow{F_{csa}} \quad = \quad (\upsilon(G_r) - \upsilon(G_l))\hat{\overrightarrow{s_r}}$$

Summation

All three *dynamic adaptation* forces are added with various coefficients to get a final pathfinding force

$$\overrightarrow{F_{pathfinding}} \quad = \quad \alpha_{cm}\overrightarrow{F_{cm}} + \alpha_{doa}\overrightarrow{F_{doa}} + \alpha_{csa}\overrightarrow{F_{csa}}$$

In practice, this force is then converted to yaw, pitch, bank, and thrust commands to try achieve a corresponding horizontal acceleration.

Restricted area avoidance

Restricted areas are permanently stored in an onboard database and therefore avoidance can be planned and optimal paths computed onboard drones. We propose here the use of the classic artificial intelligence $A*$ algorithm (Hart, Nilsson, and Raphael 1968).

$A*$ is a graph search algorithm. It computes a list of nodes to visit to maximize a given heuristic while going from a start node to a target node and avoiding unwanted nodes. The path computed can be optimal (node-wise), and, if the graph is not too big, the computation is relatively inexpensive. A pseudocode listing of $A*$ is presented in algorithm 6.

If we divide the SAR operations area in grid cells, each cell representing a node, we can use $A*$ to compute optimal paths between two cells, by using an euclidian distance heuristic (which, as a side note, is an admissible heuristic ensuring optimality), and tagging certain cells as restricted (or not usable). In practice, the process is to tag all cells containing a part of a restricted area as unusable.

The standard version of $A*$ only takes into account neighbor nodes of each node to perform its computations, and visiting each in sequence will look unnatural ("jerky" path with multiple turns instead of straight lines), as in figure 13 where nodes take the form of grid cells.

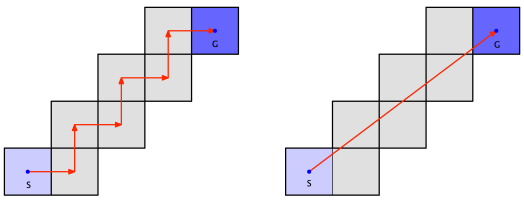


Figure 13: Paths without / with post-smoothing

Post-smoothing Post-smoothing is the process of joining two cells that are in line of sight with a straight line. By reducing unnecessary intermediate waypoints, it reduces the distance between two cell points while also saving battery by incurring less maneuvers. It can be computed in a number of ways; here we'll use a recursive iteration of the Bresenham algorithm (Bresenham 1965) to smoothen computed paths. Bresenham's algorithm is well known in the field of computer graphics: it is a line drawing algorithm that determines which cells to select in a grid (a screen grid for example) to obtain as close an approximation of a straight line as possible between two points.

The Bresenham algorithm's pseudocode version is presented in algorithm 7.

Here, we use Bresenham's algorithm to compute a *line of sight* between two points, in relation to restricted areas. Cells marked by Bresenham's algorithm as being on a straight line between two points A and B are checked to see if they are restricted. If a marked cell is restricted, then there is no line of sight between A and B. If no cell is restricted, then there is a line of sight between A and B and the path between A and B is simplified by removing all intermediate cells in between.

The overall logic is listed in algorithm 3.

Flight plan generation Overall, combining $A*$ and post-smoothing produces a list of nodes (which are grid cells in the present case) that have to be visited in sequence by a drone to go from a start position to a goal position. However, dynamic adaptation pathfinding is based on targeting a given *position* T.

Therefore, lists of grid cells to visit are converted to positional data by computing the position of each grid cell's center, and stored in a each drone's flight plan memory. This flight plan memory is actioned by each drone in the following manner: as soon as a target T is reached, it is removed from the flight plan sequence, and the first remaining target becomes T. If the flight plan is empty, the drone stops.

Algorithm 3 Bresenham-based post-smoothing

```
function LineOfSight(node A, node B)
    for each N ∈ Bresenham(A, B) do
        if restricted(N) then
            return False
        end if
    end for each
    return True
end function

function RecursiveSearch(node A, node B)
    if A = B then
        return B
    end if
    if preceding(B) = A then
        return B
    end if
    if ¬ LineOfSight(A, B) then
        return RecursiveSearch(A, preceding(B))
    end if
end function

function PostSmoothing(list P)
    if ¬ empty(P) then
        list L ← new list
        node S ← first(P)
        L ← add S
        while S ≠ last(P) do
            S ← RecursiveSearch(S, last(P))
            L ← add S
        end while
        return L
    end if
    return P
end function
```

3.2.4 Intentions and policies

In order to be autonomous in their environment, cognitive drones must exhibit a certain set of behaviors: they need a set of intentions (possibly given by the operator), and ways of successfully achieving these (by either designing a plan or following a procedure). Plan generation is powerful but as with all things subject to learning, mistakes and approximations are part of the process (things rarely unroll exactly as planned!); and in the present context we want drones to be immediately operational and efficient in performing SAR operations, without providing *training*. The procedural route seems therefore more appropriate: a set of behavioral policies is thereby conceived. Policies are actioned depending on the drone (or the operator's) intentions.

Intentions

Drones keep track of a duality of intentions: an *internal intention* that can be modified by the drone itself depending on the situation, and an *external intention* (operator intention) which is only assignable and modifiable by the swarm operator. Drones' internal intentions are initially set to match any assigned external intentions, but are subsequently modified as necessary to manage im-

peratives such as battery charging for example.

As an example is sometimes worth a thousand words, let's imagine the swarm operator has assigned the external intention DEPLOY to drones. When they reach the associated assigned position, their internal intention becomes STANDBY. After a while, a certain number of drones assess that their current position and battery charge level (both provided by individual cognition systems) is making energy requirements a concern and that recharging is soon to become critical. They automatically assign themselves a RECHARGE internal intention, and start searching for the nearest landing pad. Once charging is done, they again modify their internal intention to DEPLOY to move back to the location they left, and finally revert to a STANDBY internal intention and are now again on par with external (operator) intentions.

Internal intentions

There are 5 internal intentions:

- STANDBY - stay immobile at the current location
- DEPLOY - move to a specified location
- EXPLORE - perform a stochastic exploration of surroundings
- FOLLOW - follow another entity
- RECHARGE - find and move to the nearest available landing pad to recharge

External (operator) intentions

External intentions are assignable by the swarm operator. There are 2 external intentions:

- DEPLOY - go to a location, by using in sequence the DEPLOY and STANDBY internal intentions
- RESCUE - initiate internal intention EXPLORE; allow the swarm to ask queries about what to search for, then FOLLOW matching entities to rescue while broadcasting a rescue intervention request

Policies

Policies are pre-wired, "instinctive" list of actions that depend on intentions. Just as living beings are equipped with various life-saving reflexes (such as breathing for example), drones are provided with policies enabling them to "thrive" in their environment.

For each internal intention, the associated policy is described by the following flow charts (figures 14, 15, 16, 17, 18).

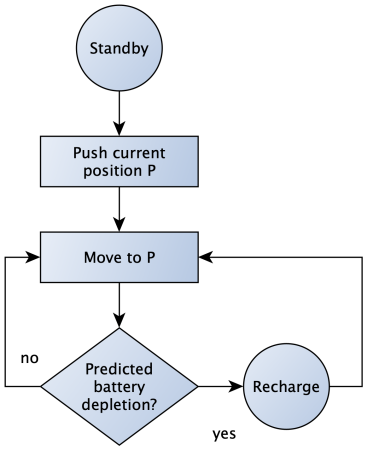


Figure 14: Standby

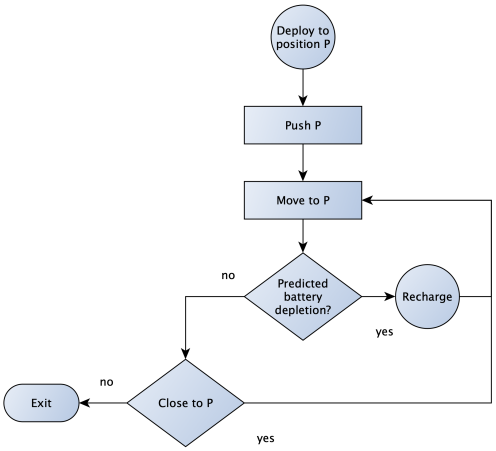


Figure 15: Deploy

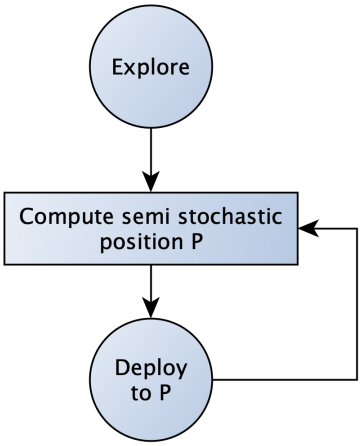


Figure 16: Explore

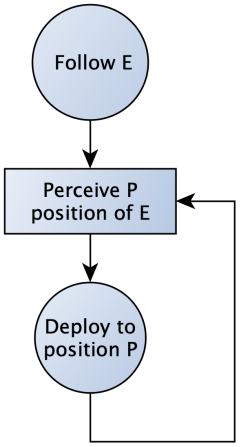


Figure 17: Follow

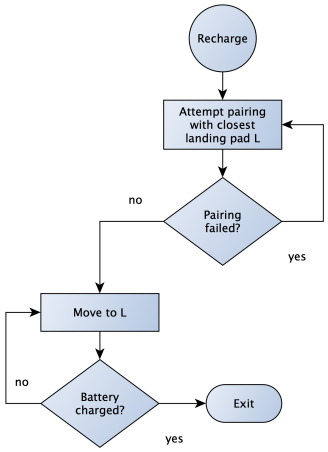


Figure 18: RECHARGE

Exploration policy Exploration is one of the principal functions of SAR drones, to find *who* to rescue and *where*. Here, we propose the use of a semi-stochastic algorithm (semi stochastic in the sense that it won't send drones back on their own recent "footsteps" nor in non-navigable areas of the map).

First, as it is random, it is impossible to know in advance its next target location and thus cannot be tricked by an adversarial trying to avoid detection. Although this sounds inutile in the context of SAR (who would not want to be rescued?), it is actually useful as living beings to rescue could potentially be avoiding detection *without wanting to* by moving in a given direction, and finding their movements match those of a predetermined drone search pattern with a delay. That would leave drones completely blind to their presence, whereas it is statistically close to impossible to match random patterns hence semi-stochastic location assignments prevent this unwanted behavior.

As with any search algorithm confronted to targets moving unpredictably, it can either be *lucky* or *unlucky* in its path selection. Also, it is *not optimal* in terms of area covered, as it doesn't systematically target every area of the map in sequence, but by having a better chance of finding moving objects than a methodical approach it compensates for that; and finally, it is based on the hypothesis that all areas of the map have the exact same probability of entailing rescue operations.

The semi-stochastic exploration algorithm avoids locations that:

- are in the drone's spatial memory
- are inside restricted areas

- are not in line of sight (obstructed by perceived obstacles residing in the drone's "mental model")

We want to make sure that barring obstacles and restricted areas, drones are guaranteed to find an unvisited location after a number of tries. If r is the distance to the next location, for this to happen we need the area of the circle with radius r to be greater than the area covered by the n locations in spatial memory. However, in the worst-case scenario, a drone can be in a corner of the map and only have a quarter of this circle available for its next location. Thus, if a is the area of a location in spatial memory, we must have:

$$\frac{\pi r^2}{4} > a\,n$$

Which has a positive solution:

$$r > 2\,\frac{\sqrt{a\,n}}{\sqrt{\pi}}$$

If we set $m = 2\,\frac{\sqrt{a\,n}}{\sqrt{\pi}}$, with σ_{md} the minimum desired distance to the next position, γ_{dc} a direction continuity coefficient such that $0 < \gamma_{dc} < 1$, k a reach coefficient ($k > 1$) and p the drone's position, the actual semi-stochastic exploration algorithm is listed in algorithm 4. Functions RAND() and GAUSSIANRAND() respectively return random numbers between 0 and 1 and gaussian-distributed random numbers (normally distributed with mean 0 and standard deviation 1). Gaussian random numbers and the related coefficient γ_{dc} are used to statistically lessen significant direction changes.

The probability $P(k\,m\,\text{RAND}() > m)$ is $\frac{k-1}{k}$: for example with $k = 4$, in the worst possible case, every $k\,m\,\text{RAND}()$ call has 3 chances out of 4 of returning a guaranteed acceptable next position, barring obstacles. Lower k values favor more progressive, local exploration whereas higher k values favor extra reach and can result in drones traveling greater distances from one position to another.

Battery charging policy Battery charging is one of the few "survival instincts" of drones aside from dynamic pathfinding. The question of when to recharge is not trivial at all, as an error will force an emergency landing in a disaster area, and the drone will certainly be lost. However, initiating the recharge process too quickly will also incur many return flights to the nearest landing pads and this will be very costly operationally (while recharging, drones cease SAR duties).

Algorithm 4 Semi-stochastic exploration algorithm

```
function RandomPosition(drone D, tries t)
    if t < 0 then
        return p
    end if
    d ← σ_md + k m rand()
    θ ← yaw(D) + π γ_dc gaussianRand()
    p_t = p + d{cos θ, sin θ, 0}
    if p_t ∈ SpatialMemory(D) or restricted(p_t) or ¬sight(p_t) then
        return RandomPosition(D, t − 1)
    else
        return p_t
    end if
end function
```

In other words, the resulting algorithm must be a little conservative, but not too much.

Our proposed algorithm predicts battery level on arrival (B_p) at the current nearest landing pad L_n, using the current battery level B_c, the average battery depletion rate Δ_b, the drone's maximum speed s_m and the current drone position p:

$$B_p \quad = \quad B_c - \frac{d(p, L_n)}{\sigma \, s_m} \Delta_b$$

σ is a conservative positive factor ($\sigma < 0.1$) as the computed distance $d(p, L_n)$ does not account for obstacles.

In practice, when the resulting battery level B_p is under a predefined threshold, the drone changes its internal intention to Recharge and temporarily ceases SAR operations.

It then tries to pair itself with the closest available landing pad (landing pad status is known via radio) by broadcasting a pairing request that includes the landing pad's identification number. If the request is unsuccessful, it tries another landing pad (the next closest): as there is at least one landing pad for each drone, the request is guaranteed to succeed rapidly even if all drones decide to recharge simultaneously.

3.3 Landing pads

Landing pads have two physical functionalities: 1) offer drones a safe landing area and 2) perform bilateral communications with the swarm of drones and with the operators' station. They are, in fact, the radio transmission relay between drones and an operator station.

Additionally, they manage incoming recharge requests through a pairing protocol. Here is what happens when a landing pad receives a drone's pairing request:

- if the landing pad is not paired, the query is accepted and a pairing acceptation message including the accepted drone's identification number is broadcast
- otherwise, the landing pad does not reply

That way, every drone is informed of its query response. If drones send simultaneous requests to the same landing pad, only one is chosen and its identification number broadcast, and others are immediately informed of their request refusal as the broadcast identification number does not match theirs.

3.4 Operator stations

Operator stations are the human-swarm interface. They are used to store all incoming swarm queries, and present them to the operator(s) using a human-readable format (as soon as a query is answered, identical queries in memory are discarded), and also to receive all drone rescue intervention requests and dispatch them to the rescue intervention team upon an eventual positive visual confirmation (via the drone's camera).

Chapter 4

Implementation

In order to test the ideas presented in this Master's dissertation, a software SAR simulation has been developed. It consists in two subparts: a generic UAVs simulation framework and an iteration of this framework applied to a SAR operations environment using a cognitive UAV swarm; but many others could also be thought of and implemented.

The codebase is designed with modularity in mind: any kind of object with its own characteristics, cognitive abilities, physics, or world interactions is easy to add to the simulation.

Other iterations of the simulation could, for example, introduce new drone models that have completely different purposes and governing algorithms, be it for learning, pathfinding, or anything conceivable; other world objects such as ground vehicles, ships, or new kinds of obstacles or natural features such as mountains, lakes, seas, etc.

The codebase is hosted online and accessible on request [1]. It consists of a set of about 200 Java classes and 7000+ lines of code.

4.1 Software platform

For a simulation environment to be broadly useful, it must at least:

- be simply deployable on a variety of environments (Windows, MacOS, Linux)
- be fast enough to function properly on a modern laptop
- be modular enough to allow for future use cases

[1] https://github.com/gvoirin/airswarm

Language-wise, it was decided to make use of *Java* and the *Valhalla* future evolution of the JDK[2]. The simulation environment is designed to run on any *Valhalla*-compatible JVM [3]: the best of both worlds regarding speed and platform portability. For a detailed explanation about technical choices, please see appendix A.1.

Three open-source libraries were also selected for inclusion in this project: the minimal-json[4] library to read JSON scenario files, the JavaFX[5] library to take care of the two and three-dimensional displays software layer, and the Weka[6] library for its decision tree classifiers.

4.2 Generic simulation world

The generic simulation world is graphically delimited by a black boundary and governed by the same laws we are accustomed to: i.e. gravity, air resistance, time, etc. It is updated in real-time, although time can be slowed down using the appropriate control (see section 4.3).

4.2.1 Content

The simulation world can contain an arbitrary number of components (overall performance being the only limit) each having its own three-dimensional position and associated three-dimensional representation in the main view. Also, the world is backed by a grid array that can serve purposes such as $A*$ computations or information storage. Every three-dimensional position within the world's boundary has corresponding two-dimensional grid cell coordinate values as in figure 19.

4.2.2 Scenarios

Scenarios represent objects' states (positions, features, etc.) in the simulated world, at time $t = 0$.

The JSON (*JSON* 2020) file format is an ideal storage medium for scenario files, as it is easily readable and self-explanatory while having enough data structure wrappers (objects, arrays...) to enable hierarchical storage: it is used for the simulation iteration of this Master's dissertation. An excerpt of a scenario file (taken from the cognitive search and rescue swarm of drones system), is presented in figure 20.

[2] Java Development Kit
[3] Windows, MacOS or Linux versions: https://jdk.java.net/valhalla/
[4] https://github.com/ralfstx/minimal-json
[5] https://openjfx.io
[6] https://github.com/Waikato/weka-trunk

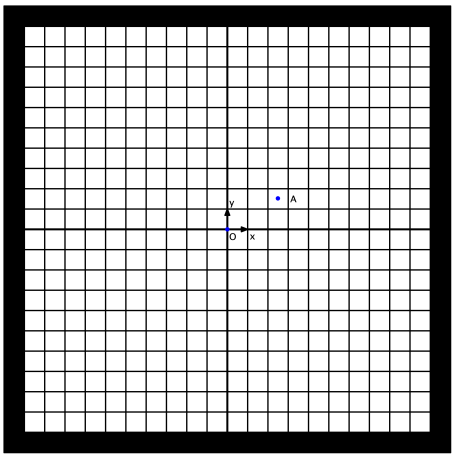


Figure 19: Dual coordinate system in the simulation world

Points O and A are defined by 3-D world coordinates $(0, 0, 0)$ and $(25, 15, 10)$ in the above schematic (Oz is not pictured as the eye is facing down) and are also part of grid cells $(10, 10)$ and $(12, 11)$

```
...
  "drones": [
    {
      "dimensions": [
        1,
        1
      ],
      "max_thrust": 15,
      "mass": 1,
      "position": [
        150,
        150
      ]
    }
  ],
...
```

Figure 20: Excerpt of a JSON scenario file

4.2.3 Physics engine

In order to compute realistic object movements, our simulator is equipped with a physics engine. Newton's laws of motion (*Newton's Laws of Motion* 2020) are applied to a subset of objects in the simulator, called "physical" objects to distinguish them from the remaining "virtual" objects on which these laws do not apply.

In Newton's physics, all objects are subject to a set of forces which can be aggregated as a single summation force $\overrightarrow{F}$ (i.e. $\overrightarrow{F}$ is the sum of forces applied

to the object in N). The set includes by default objects' own weight computed as $\overrightarrow{w} = m.\overrightarrow{g}$ and whose norm is expressed in N ($1N = 1kg.m.s^{-2}$), with m the object mass in kg and $\|\overrightarrow{g}\| = 9.80665m.s^{-2}$.

Using Newton's laws, we have:

$$\overrightarrow{F} = m.\overrightarrow{a}$$

Hence,

$$\overrightarrow{a} = \frac{\overrightarrow{F}}{m}$$

Every computation iteration, a time difference elapsed since the last computation iteration is retrieved (in floating-point number of seconds) and referred to as dt.

New speeds and positions are then computed as such:

$$\overrightarrow{s} = \overrightarrow{s} + \overrightarrow{a}.dt$$

$$p = p + \overrightarrow{s}.dt$$

Physical objects can additionally be subjected to air resistance or drag (*Drag (Physics)* 2020), in which case a drag force directly opposed to the speed vector is automatically added to an object's sum of forces $\overrightarrow{F}$. Its norm is computed as follows:

$$R = \frac{1}{2}\rho A C_d \|\overrightarrow{s}\|^2 \tag{4.1}$$

With $\rho = 1.225kg.m^{-3}$ the density of air, C_d the drag coefficient and A the exposed surface variable for each object type.

Drone physics

Drones are quadcopters and their specific physics complement the simulation framework's physics engine. As other physical objects, they are subject to weight and air resistance forces.

Level flight Furthermore, as they operate in level flight, their sum of forces is vertically nil (otherwise a climb/descend force will incur vertical movement). If we set aside air resistance (which has no vertical component as movement takes place on a horizontal plane), we get the vertical constraints system schematic displayed in figure 21. The following are thus derived from level flight con-

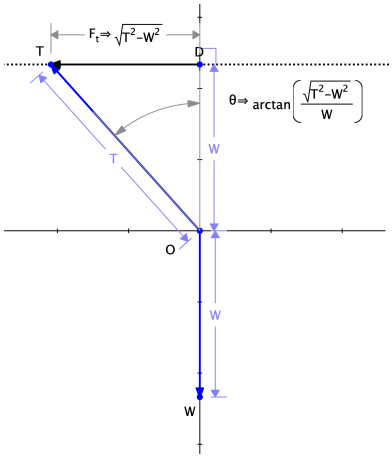


Figure 21: Physics of level flight

O is the drone's center of gravity, θ the drone's pitch angle, $\overrightarrow{OT}$ represent the thrust vector $\overrightarrow{T}$ and $\overrightarrow{OW}$ the weight vector $\overrightarrow{w}$. F_t is the forward thrust component

straints:

$$\begin{aligned} W &= T\cos(\theta) \\ F_t &= \sqrt{T^2 - W^2} \end{aligned}$$

Obviously, $T \geq W$ otherwise the drone descends.

Dynamic thrust Depending on the drone's speed and position in regards to relative airflow, propeller thrust efficiency varies. Requested thrust is therefore corrected to dynamic thrust: drones' four propellers are assimilated to a single one whose center lies at the drone's center of gravity, and the thrust correction force $\overrightarrow{D}$ is computed depending on this propeller's angle to the relative air flow. Experimental testing has demonstrated a near-linearity of the obtained thrust / demanded thrust ratio in regards to relative speed of propellers to surrounding air (jburgoyn 2014; Staples 2019) so we'll use this good enough approximation here.

$$\overrightarrow{D} = -\eta \hat{\overrightarrow{T}} \| \overrightarrow{s} \| \frac{\sqrt{T^2 - W^2}}{T} \cos(\widehat{\overrightarrow{s}, \overrightarrow{T}})$$

The formula can be demonstrated with usual trigonometry. $\overrightarrow{T}$ is the thrust produced by propellers in still air (as demanded by the drone's software pilot) expressed in N, θ the pitch angle, $\overrightarrow{w}$ the weight, η a predefined penalty factor, and $\widehat{\overrightarrow{s}, \overrightarrow{T}}$ is the angle between the quadcopter's speed and requested thrust vector in a horizontal plane.

The new thrust force vector after correction is then

$$\overrightarrow{T_c} = \overrightarrow{T} + \overrightarrow{D}$$

However, the corrected thrust vector must also maintain level flight constraints and this incurs a change in pitch angle from θ to θ' (as $W = T_c \cos(\theta')$). This is a differential equation: dynamic thrust modifications incur changes in pitch angle to maintain vertical flight constraints, but changes in pitch angle also modify dynamic thrust values. However, the variation of dynamic thrust from pitch angle θ to θ' is usually minimal and the associated numerical error is accepted here.

The air resistance and weight forces are then added to $\overrightarrow{T_c}$ (see section 4.2.3) to obtain the final force $\overrightarrow{F}$ (which must have no vertical component if computations are right, due to the level flight constraints).

Newton's laws of physics are then applied to deduce the resulting acceleration $\overrightarrow{a}$ and from there speeds and positions are updated.

4.3 User interface

The simulation user interface consists of a single window, subdivided into distinct panes (see figure 22).

The **main view pane** displays a three-dimensional bird's eye view of the world. It is possible to zoom in and out, click into, or translate and rotate the viewpoint using the mouse.

On the top right, the **individual cognition pane** represents the drone's mental model from a bird's eye perspective, with symbols and colors the same as in the main view. Next to it, the **collective speed memory pane** displays collective average speed memory in the form of a heatmap, again from a bird's eye perspective (the whole map is represented).

Just under these two panes, the **mouse action pane** offers four selectable options: *position*, to point to a position in the simulated world via a click inside

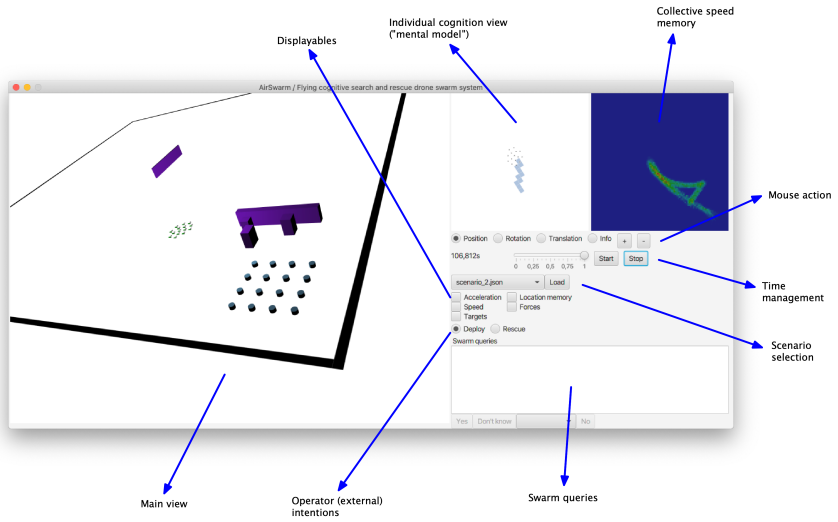


Figure 22: Simulation user interface window

the main view, *translation*, to move the camera (eye) left/right or up/down in the world, *rotation*, to rotate the camera angle, and *info*, to select an object in the main view. Additionally, this pane includes a set of zoom buttons to zoom in and out of the main view (zoom is multiplied/divided respectively by a scale factor of 1.1).

Under the mouse action pane, the **time management pane** enables control of simulation time. It includes a simulation time display as well as start/stop buttons and a slow-motion controller). The **scenario selection pane** is a drop-down list used to select and load a scenario file from disk. The **operator (external) intentions pane** is used to send external (operator) intentions to the swarm: DEPLOY or RESCUE.

The **swarm queries pane** simulates operator stations querying: swarm queries are accessed there in an unformatted manner. They can be replied to asynchronously (they are non-blocking: the simulation continues to run with unanswered queries).

4.4 SAR simulation world

The SAR simulation world consists of five types of elements: cognitive drones, landing pads, obstacles, restricted areas, and other entities. At $t = 0$, drones have no knowledge of obstacles and other entities.

4.4.1 Cognitive drones

Cognitive drones are implemented using the algorithms presented in chapter 3.

In particular, the horizontal pathfinding force $\overrightarrow{F_{pathfinding}}$ is computed for each drone, then converted to a desired thrust vector using the laws of level flight ($\overrightarrow{F_{pathfinding}}$ gives the direction of the required forward thrust component as in section 4.2.3), which is itself associated with yaw, pitch, bank, and thrust values.

Also, collective speed memory diffusion (cf. section 3.2.2) being quite computer intensive (a simulation world can be divided in millions of grid cells), it is massively parallelized and computed in small batches at every time step dt.

Cognitive drones representations are displayed in figure 23. The green bar underneath displays battery level. Instantaneous battery depletion is computed proportionally to the current thrust vector norm. The image on the far right shows drones above landing pads.

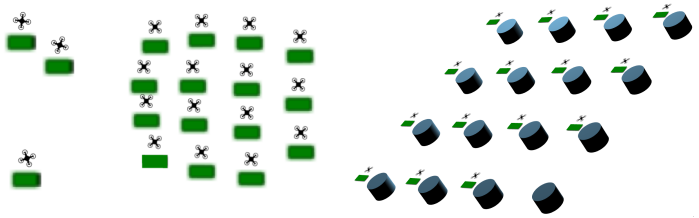


Figure 23: Cognitive drones representations

For this simulation's purposes, a set of implementation scalars has been chosen. They are listed in table 1 with their reference section.

$r = 50$	section 2	$\eta = 0.001\frac{size(grid)}{\sqrt{size(grid)}}$	section 3.2.2
$\delta_{wm} = 60s$	section 3.2.2	$\rho_c = 20m$	section 3.2.3
$\delta_{acc} = 300s$	section 3.2.2	$\rho_s = 10m$	section 3.2.3
$\delta_a = 10s$	section 3.2.2	$\rho_a = 10m$	section 3.2.3
$\alpha_c = 0.2$	section 3.2.3	$l = 2m$	section 3.2.3
$\alpha_s = 5$	section 3.2.3	$\varphi = 100$	section 3.2.3
$\alpha_a = 1$	section 3.2.3	$\gamma_a = 1.25$	section 3.2.3
$\alpha_{em} = 50$	section 3.2.3	$k = 2$	section 3.2.3
$\alpha_{cm} = \frac{1}{3}$	section 3.2.3	$\alpha_{doa} = 1$	section 3.2.3
$\alpha_{csa} = 1$	section 3.2.3	$\sigma = 0.1$	section 3.2.4
$\gamma_{dc} = 0.75$	section 3.2.4	$k = 4$	section 3.2.4

Table 1: Implementation scalars

4.4.2 Landing pads

Landing pads' inner works (in particular, the drone pairing algorithm) are presented in chapter 3. In the present implementation, drones just need to hover above landing pads to start recharging.

They are represented by cyan cylindrical volumes, as in figure 24.

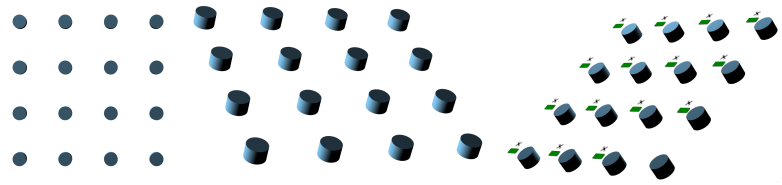


Figure 24: Landing pads representations

4.4.3 Obstacles

Obstacles take various purple three-dimensional rectangular shapes in the present implementation, as in figure 25.

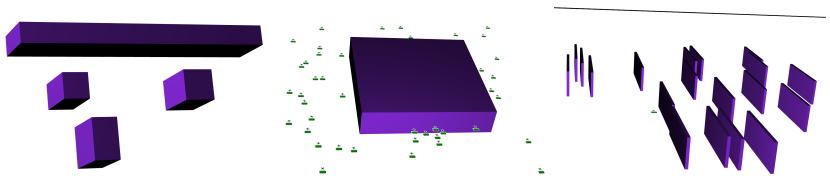


Figure 25: Obstacles representations

4.4.4 Restricted areas

Restricted areas surfaces are depicted in grey as in figure 26, and consist of ellipse and rectangle-like figures.

Figure 26: Restricted areas representations

4.4.5 Rescuable entities

Rescuable entities can take various colors, sizes or shapes as in figure 27: they are only symbolic representations, used as abstractions to any kind of real object or living being. They can move in the simulated world if required, and are able to avoid dynamic obstacles. They are allowed to enter drone-restricted areas, and can trigger a rescue intervention if their "fuzzy feature vectors" (see section 3.2.2) match certain characteristics induced from the swarm-operator relationship.

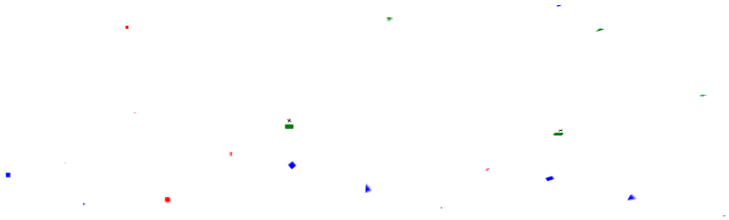


Figure 27: Rescuable entities representations

Chapter 5

Evaluation and testing

Our implementation of a simulation environment, described previously, now enables us to test various aspect of our cognitive swarm of drones system, through a set of challenges. These will test swarm units both individually and collectively.

5.1 Pathfinding

Pathfinding is, by far, the main functionality of exploration drones. In our approach, pathfinding takes its inputs from cognition, but often, as drones are exploring, they are yet to acquire knowledge of their environment. We'll therefore test both pathfinding *without* environment knowledge (in which case cognition has close to zero immediate effects), and *with* environment knowledge (where cognition-provided information should enable cleverer paths). Also, algorithms chosen for *restricted areas* pathfinding will be evaluated.

5.1.1 Zero-knowledge dynamic pathfinding

Zero-knowledge pathfinding is crucial: if not implemented properly, drones won't be able to explore all areas of the map and thus learn from them: poor zero-knowledge pathfinding can in fact severely impact drone cognitive abilities. Not to mention the fact that, in a SAR environment, it is clearly detrimental not to be able to explore all parts of the operational area.

Single drone

Our first challenge consists in putting a single drone in front of a complicated, unseen-before obstacle field (no learning of the environment has been allowed).

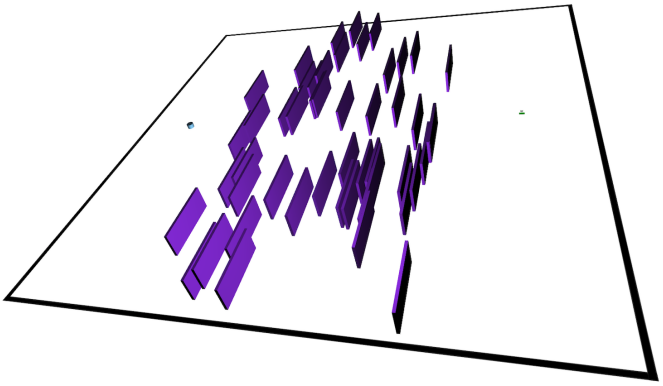


Figure 28: Single drone zero-knowledge pathfinding challenge

Its aim is to go to a landing pad on the other side of the field, at a distance of 150m. Obstacles are wide and numerous, and completely prevent a visual sight on the target as in figure 28. Dynamic obstacle avoidance (see section 3.2.3) is put to a serious test here: we expect the drone find its way, possibly with difficulty (via a non-optimal path), but successfully.

Step by step progress is shown in figure 29. We can observe that the drone progresses with general ease and constancy in a quite difficult pathfinding environment. Between $t = 25s$ and $t = 42s$, the drone enters a dead end area due to a combination of speed and inertia, but quickly turns back thanks to static obstacle repelling forces (cf. section 3.2.3) and the escape maneuver (cf. section 3.2.3), and eventually finds its way out.

Overall, its path is smooth and few slowdowns are observed although the close proximity of obstacles entails moderate speed. Our sum-of-forces approach to dynamic pathfinding is an overall success here.

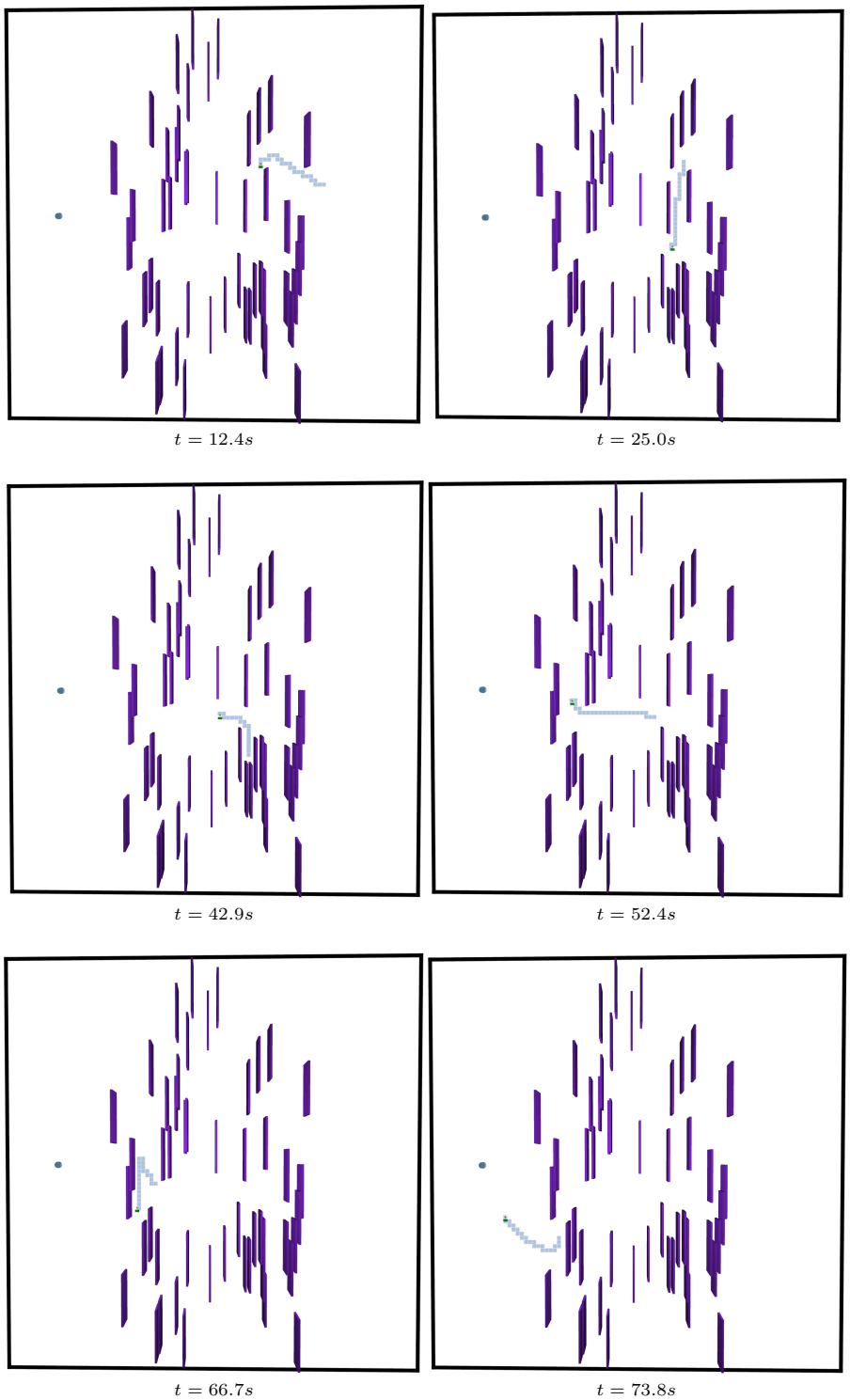


Figure 29: Single drone zero-knowledge pathfinding progress

Multiple drones

For our second challenge we set up the same type of environment, albeit this time we reduce the number of obstacles and test a flock of 20 drones having the same intention (i.e. to reach the landing pad on the other side of an obstacle field). As a drone swarm covers more area, it's a very difficult challenge as units have to inter-organize in order to find passages through narrow gaps, while avoiding each other. The challenge visual overview is presented in figure 30.

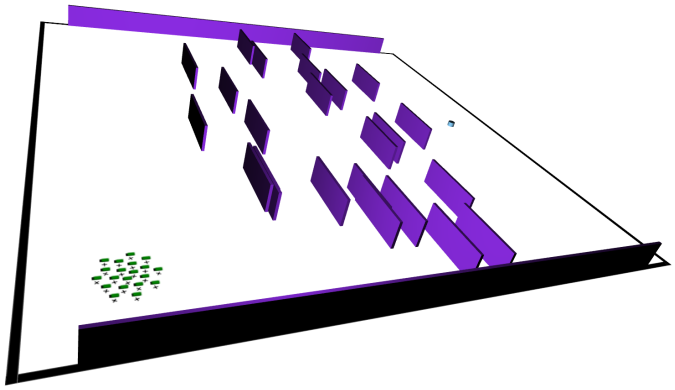


Figure 30: Multiple drone zero-knowledge pathfinding challenge

We expect drones to find their way, by eventually splitting into sub-groups or following one another through narrow gaps (as collective movement forces issued from the boids algorithm described in section 3.2.3 favor group cohesion).

Figure 31 shows a timeline of the drones' progress. As there are less obstacles and thus more space in between, the first drones of the swarm are able to pickup speed and reach their target quickly. At narrow gaps, slowdowns are observed for the other drones, and one even interestingly reverts to an escape maneuver (cf. section 3.2.3) to find its way. However, all drones eventually reach their target successfully.

Overall, our sum-of-forces approach seems to work here as well: its relative simplicity can entail emerging group behaviors such as sub-group splitting, and even individual reactive behaviors when a unit leaves the group to find its own way after being blocked by the others for too long.

It has to be noted, though, that there are corner cases in which the escape maneuver is clearly sub-optimal: it can send drones in any direction depending on walls, and this can include a direction opposite to the target. There is certainly room for improvement on that matter. However, drones eventually reach their targets even by using the long route: this is overall a very positive outcome.

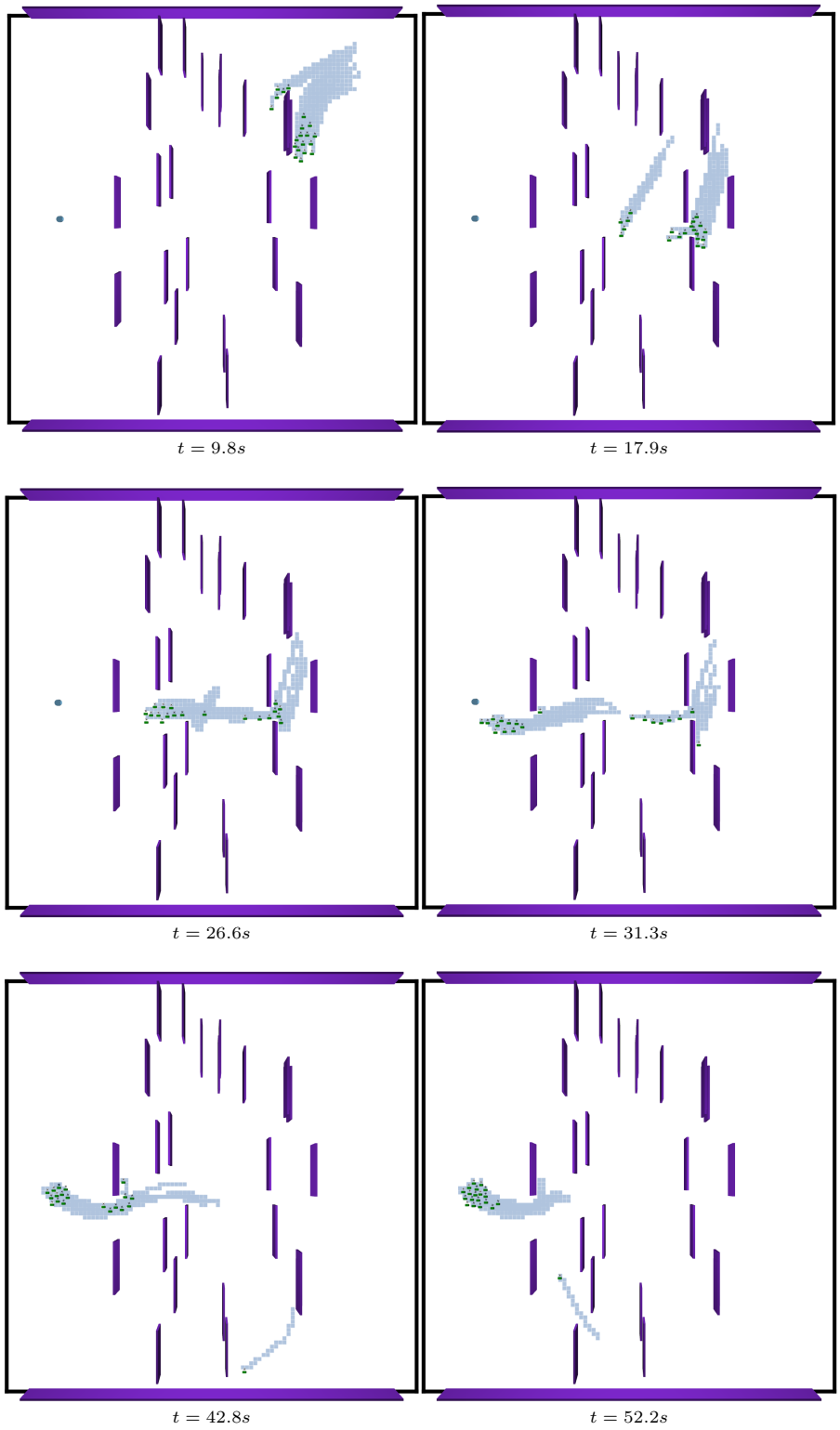


Figure 31: Multiple drone zero-knowledge pathfinding progress

5.1.2 Cognitive dynamic pathfinding

Drones constantly acquire knowledge while in their environment, even by staying static: they learn from other drones' status, or eventual unknown entities passing close by. But like many living beings, exploring is their main knowledge acquisition vector. How does this knowledge then transpire in drones' pathfinding capabilities?

Here, we setup an environment that will voluntarily entice a drone to make the wrong choice, without it knowing as it does not yet have any knowledge of its environment. But it should, thanks to its cognitive functions, be able to avoid the wrong route afterwards whereas non-cognitive drones would repeatedly fall into the "trap".

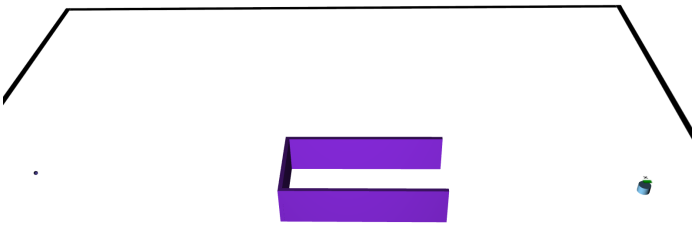


Figure 32: Cognitive test

The schematic of the concept is displayed in figure 32. A drone is on a landing pad to the right, and is requested to deploy to the left side of the map to a target depicted by a dark blue dot. However, in the middle of the map lies an obstacle (exactly in between the landing pad and target), that cannot be perceived yet as it is too far for the drone's LiDAR system in this scenario. The drone is thus expected to go straight towards the wall ahead until it realizes it is a dead end.

The interesting thing about this scenario is that it potentially puts to test the **escape maneuver** (to exit the dead end), **individual cognition** to memorize the environment once it has been discovered, **collective cognition** with speed memory to smoothen paths, and the **battery charging policy** to send the drone back to its landing pad for a recharge. To make the overall process automatic, the battery onboard the drone has a low capacity and depletes very quickly. Once the drone reaches its target on the left, it takes only a few seconds before it initiates a return to its landing pad due to low battery. Once the battery is recharged it goes back to its intended target, and so forth.

In figure 34, we can observe the following: at $t = 8.4s$ the drone sees that

the three walls around it are a dead end (also see figure 33 for a representation of the drone's current mental model status, where walls are surrounded by red borders marking attended elements). It starts an escape maneuver and then follows the top wall at $t = 24.2s$, back on its own steps. At $t = 35.7s$ it turns back again but this time decides to go above the three-wall obstacle, and reaches its target successfully at $t = 50.0s$. Once the battery charging policy decides it is time to return to the closest landing pad, it initiates its return journey at $t = 61.2s$. At $t = 71.2s$, it reaches its landing pad *with* remaining battery and starts recharging. Once the battery is fully recharged, it heads back towards its assigned target. But by $t = 86.5s$ it has done something very interesting: although its LiDAR cannot directly sense the vertical wall, its cognition (working memory specifically) has informed pathfinding about the obstacles ahead and they are avoided just as if they were sensed: the drone is using its knowledge to perform obstacle avoidance and easily avoids the "trap" this time. From here thereafter, it systematically avoids the three-wall obstacle on its journeys between landing pad and target.

After a while, collective speed memory has collected enough data (with multiple drones it goes quicker) to start its subtle job of smoothening paths, by gently pushing the drone onto its faster lanes. As it only uses a single drone's data, doing the same return journey over and over, its effect is here minor as expected. Its state at $t = 978.4s$ is displayed in figure 35, where we can see clearly the two main fast lanes (one for the forward, and one for the return journey).

So, overall, cognition does seem to improve pathfinding: there is a **clear advantage** in learning from salient parts of the environment, as this case study shows.

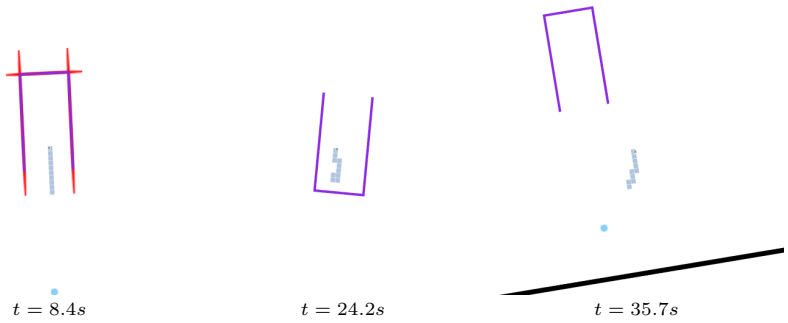


Figure 33: Cognitive test mental models

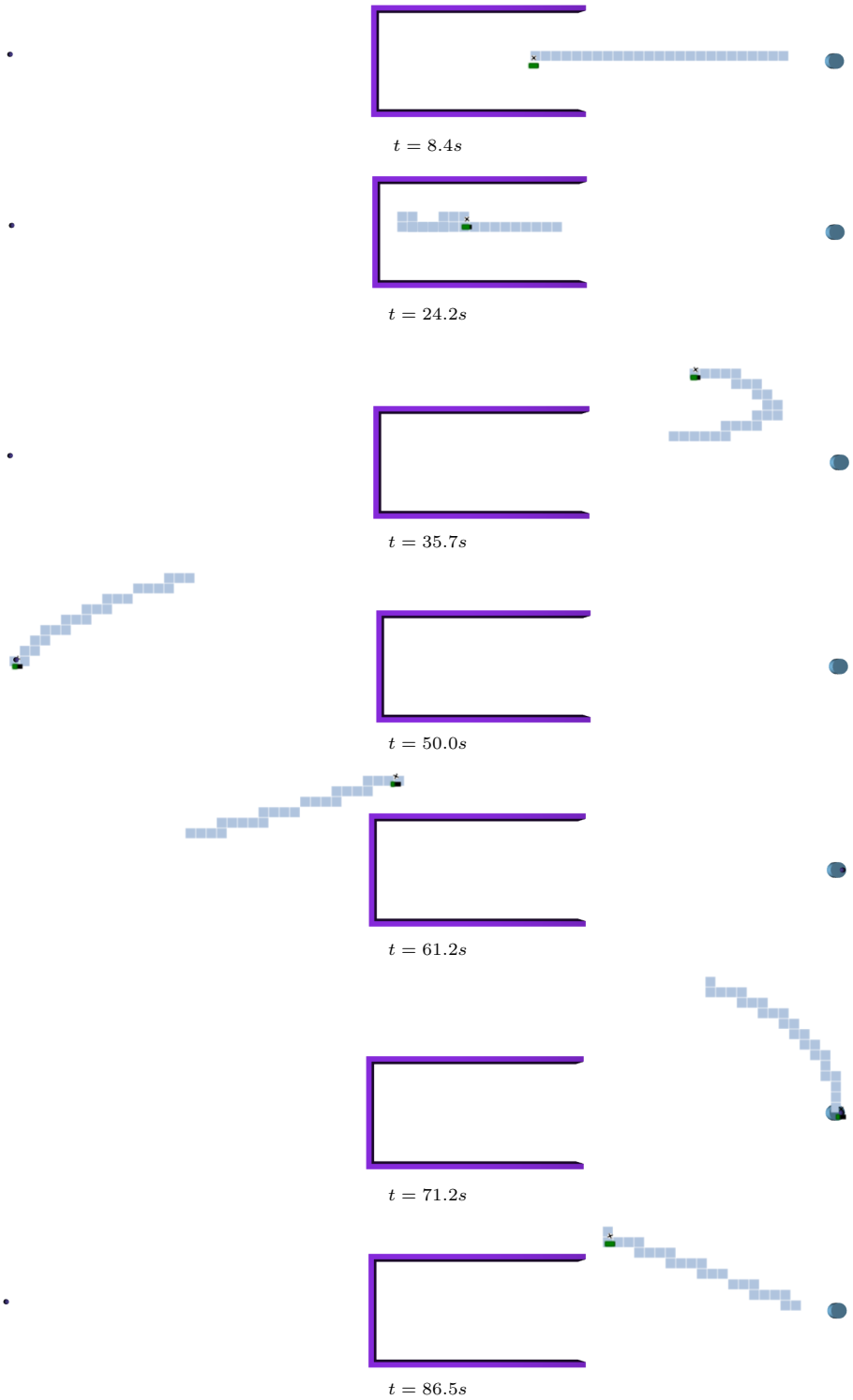
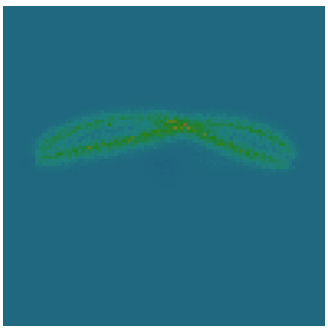


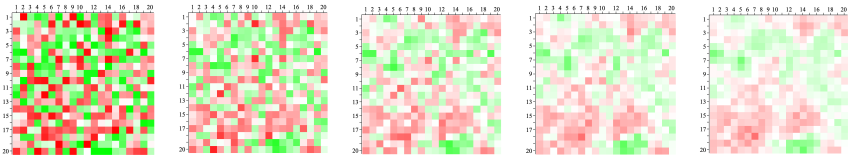
Figure 34: Cognitive test progress



$t = 978.4s$

Figure 35: Collective speed memory status

On a side note, and speaking of collective speed memory, the diffusion algorithm is as important as the storage space itself, as its aim is to propagate information in all directions and create average speed information gradients. For testing purposes, in figure 36, we apply the algorithm to a random matrix ($\eta = 0.1$). Results are satisfying: smooth gradients are created throughout the heatmap and do "generalize" information in space.



$i = 0$ $i = 2000$ $i = 4000$ $i = 6000$ $i = 8000$

Figure 36: Diffusion algorithm progress (with i the iteration count)

5.1.3 Restricted areas pathfinding

An $A*$ "test playground" is setup, with a single drone starting on a landing pad and having to navigate through a complex set of restricted areas. It steers in between restricted areas (in grey as in figure 37) and reaches its target (a dark blue dot in the bottom left, while the next intermediate waypoint is depicted by a light blue dot). The combination of $A*$ and post-smoothing works well: paths are straight with necessary turns only.

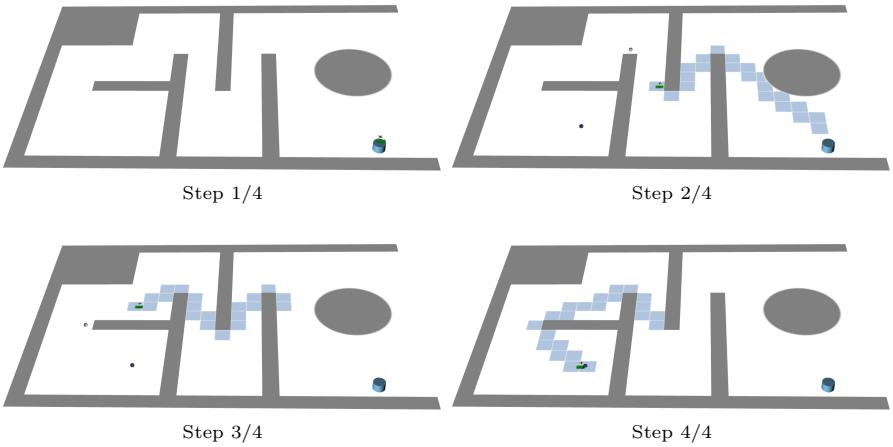


Figure 37: Restricted areas avoidance

5.2 SAR operations

The swarm of cognitive, autonomous drones performs SAR operations by 1) exploring the environment and 2) querying the operator if necessary. Exploring is done with the help of our semi stochastic algorithm, while querying involves decision trees to learn from operator responses.

5.2.1 Semi-stochastic exploration algorithm

What we want to evaluate here is whether the semi-stochastic exploration algorithm actually makes drones explore the *whole* map, even though we know this will not be performed in an optimal manner (although there is a very small probability it could be).

A 400m x 400m SAR world is setup with various SAR elements (cf. section 4.4) and 16 drones ready to go on their landing pad in a corner of the map, as in figure 38. Collective speed memory is used to verify actual exploration of the map.

Figure 39 shows map exploration progress with time. At $t = 120s$ a large part of the map is already explored (a status picture is displayed in figure 40), at $t = 240s$ an even greater part is explored, and at $t = 480s$ (8 minutes) we start to see non-navigable area contours which shows that a significant portion of the map has been explored. This is reasonably fast for an admittedly non-optimal algorithm, on a 400m x 400m area (due to simulator physics, speed and maneuverability of drones are realistic). In fact, with a $50m$ visual range (the actual visual range of drones in the present example), the whole map has been visually explored by $t = 480s$.

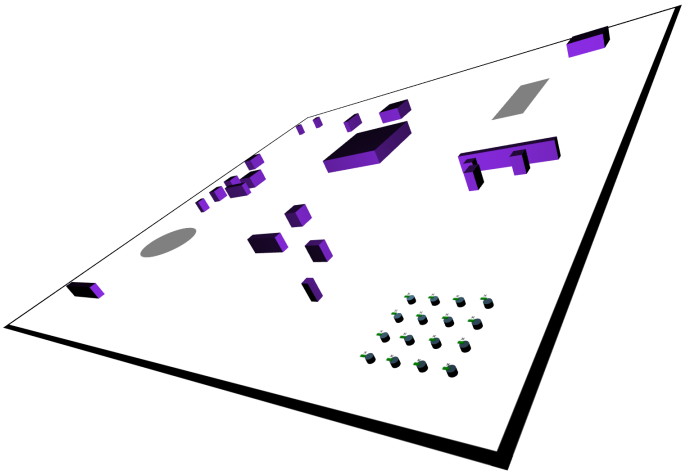


Figure 38: Semi stochastic exploration algorithm challenge map

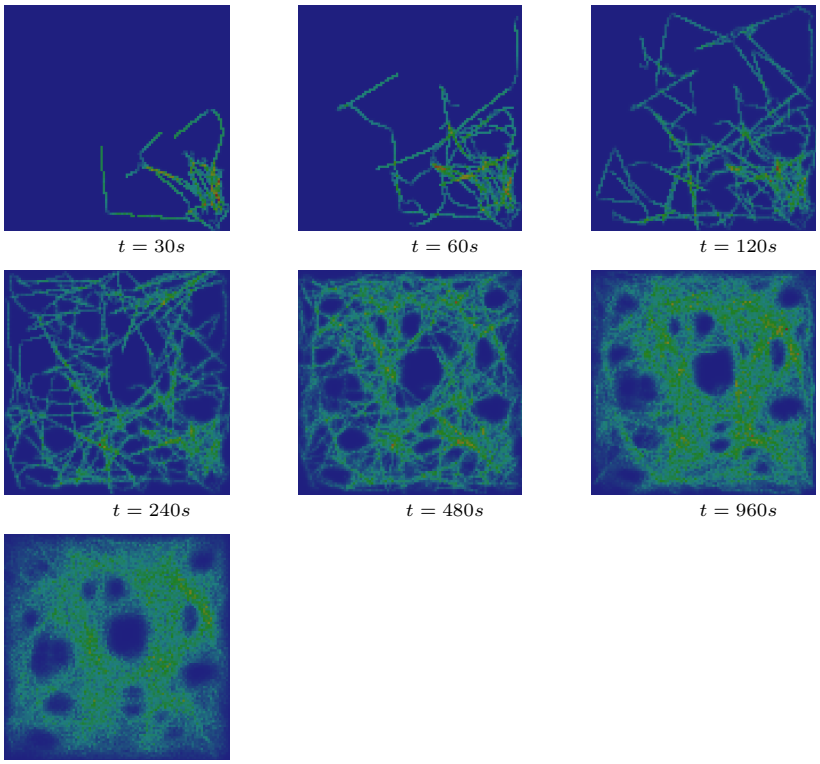


Figure 39: Semi stochastic exploration progress

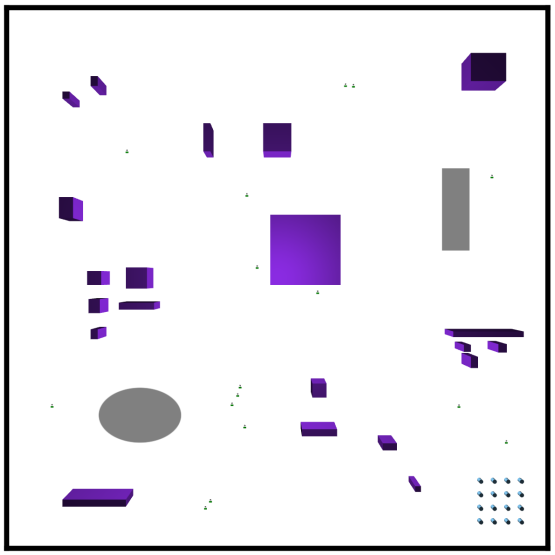


Figure 40: A general situation picture at $t = 120s$

5.2.2 Operator querying

To simulate operator querying, we consider a very restricted set of attributes: {type, attitude, size, movement}, and their associated (restricted again) fuzzy feature tags presented in table 2.

Type	**Attitude**	**Size**	**Movement**
human	aggressive	small	none
cat	lying down	large	slow
bird	waving	huge	fast
unknown	unknown	unknown	unknown

Table 2: Fuzzy feature tags set

We suppose that the operator has already given an answer to the following queries:

1) {cat, aggressive, small, slow} → {no, hint **type**}
2) {human, unknown, large, fast} → {no}
3) {unknown, unknown, small, slow} → {no}
4) {unknown, lying down, large, none} → {don't know}
5) {human, waving, unknown, slow} → {yes}

After these 5 queries, the decision tree produced by $ID3$ is displayed in figure 41.

We can see that {human, waving} already triggers an automatic rescue intervention from drones, based on operator decisions. However, due to a lack of

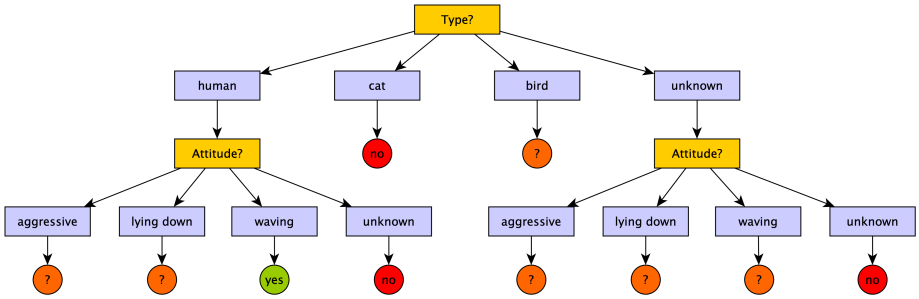


Figure 41: Decision tree after query 5

observations, numerous leaves have an uncertain outcome: the swarm continues to systematically produce observation-based queries in these branches of the tree. We can also see that operator hinting has completely excluded type "cat" from rescuing operations.

After a while, drones make other observations and the following queries are answered:

6) {bird, unknown, small, fast}→ {no, hint **type**}
7) {unknown, aggressive, small, slow} → {no}
8) {human, lying down, large, none}→ {yes}

We now have the tree displayed in figure 42.

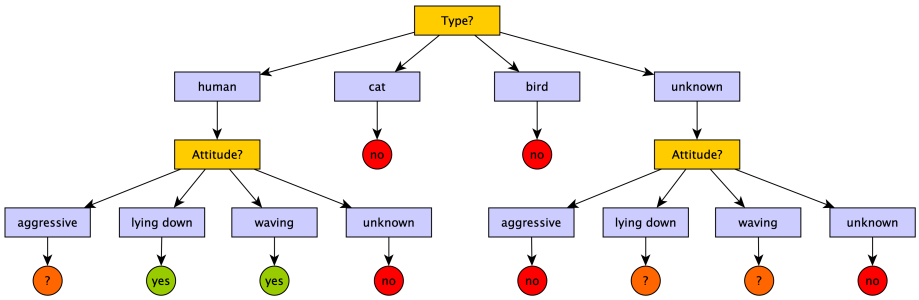


Figure 42: Decision tree after query 8

After only 8 queries, the decision tree is already very informative: {human, waving}, and {human, lying down} now trigger automated rescue interventions, cats and birds are excluded altogether as well as aggressive unknown types.

Let's now suppose that ongoing field reports inform the operator that the majority of adults waving are actually fine and just signal their ok status, whereas children waving are usually in immediate need for rescue. As a reminder, drones continue to send decidable queries based on current observations to check that nothing has changed, but on a stochastic basis to greatly limit their number

(a bias is taken that no error is made and no change is necessary). A query is considered decidable if the decision tree associates a clear outcome to it (either *yes* or *no*).

Thus, as time passes by and drones detect more and more humans, some decidable queries are re-issued, such as in the following list:

9) {human, waving, large, unknown}→ {don't know} *(was decidable)*
10) {human, waving, small, fast}→ {yes} *(was decidable)*
11) {human, waving, small, slow} → {yes} *(was decidable)*
12) {human, waving, small, unknown} → {yes}

After query 12, the actualized tree is displayed in figure 43.

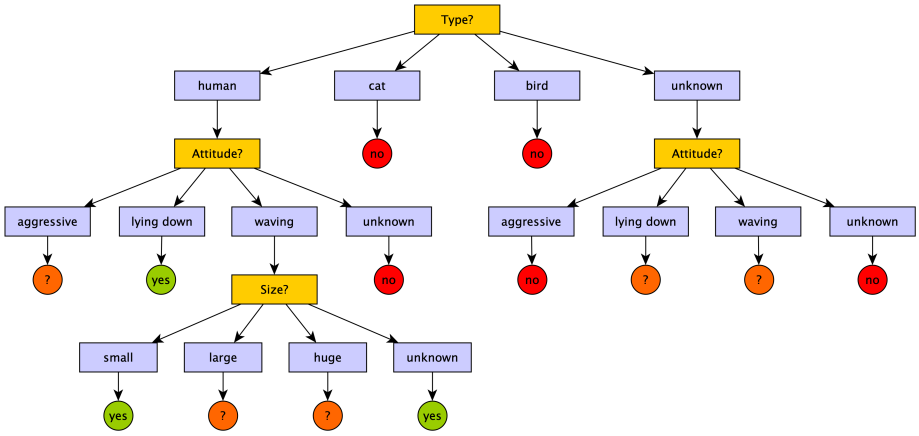


Figure 43: Decision tree after query 12

The updated tree does now make a distinction between adults and children, and only children trigger automatic rescue intervention requests, as per new operator decisions. In the same way, every new observation and associated query request updates the tree, and an operator can constantly update the swarm's knowledge state through this process: decision making is dynamic and so are updated decision trees. This system also allows for errors, as contradicting query answers are also taken into account, and the priority given to the answer with highest probability by the decision tree generation algorithm.

Chapter 6

Discussion

We have hereby attempted to achieve a symbiotic combination of simple pathfinding algorithms coupled with cognitive functionalities and artificial intelligence-borrowed tools; pathfinding providing information to cognition which in turn influences it, and cognition generating observations from its mental model which in turn trigger drone-operator queries whose replies are learned from using machine learning classification techniques.

The SAR operations context seemed like an ideal framework for such an attempt: being highly dynamic and in an unknown state, it forces swarms of drones to adapt in real time to unseen-before information and in multiple manners (via pathfinding and via observation/detection through queries).

Algorithms presented in this Master's dissertation have various roots: common sense (dynamic obstacle avoidance), natural observation (swarm intelligence with boids and ant colony-type algorithms), formal computer science ($A*$, Bresenham, or $ID3$ algorithms), or cognitive psychology principles (individual cognition algorithms), but all belong to one of the multiple facets of artificial intelligence in general.

Interestingly, cognitive drone's pathfinding algorithms, although simple individually, can collectively be considered an example of *fluid* argumentation: conflicting forces (swarm cohesion, separation, and alignment; best direction and obstacle repelling forces; individual and collective cognition-incurred forces) influence each drone's motion, just as in human argumentation, strongest sets of arguments outweigh others to offer an apparent best outcome to the problem at hand.

Although cognition is often defined through the prism what is *learn*ed, it can arguably just as well be defined by what is *forgotten*. It is striking to notice that in recent years, considerable progress in machine learning has been achieved through forgetting by, for example, using dropout techniques in neural

network training, or multiple trees based on partial data in random forest classification algorithms. By introducing timing-based information filtering through our human-inspired individual cognition system, we have actually implemented a time-based *forgetting* mechanism related to its natural counterpart. After all, cognition is also linked to perceived *saliency*, and environmental features that are continuously interacted with are more likely to be considered salient (and learned from) than, for example, the fly that just went out through the window. Here, the words *more likely* are important, as cognition in general (and human cognition in particular) is a great deal more complex than that: sometimes transient events can leave everlasting traces in memory as other cognitive processes give them crucial importance and utmost saliency.

So, does this mean the presented system is optimal? Far from it! Improvements can actually be made in all areas.

First of all, speaking of cognition, it could be more extensive in general and integrated with other functionalities. By "more extensive", we mean that it could account for more than timing-based interactions, and include levels of *emotions* for example that would influence its memorizing and attention mechanisms. Also, it could be integrated in the *planning* stage, together with $A*$ computations, battery recharging requirements, or general behavior policies to bring a component of learning to these — although this might be undesirable in some cases for uniformity reasons.

Pathfinding could also be improved: first by making algorithms fully three-dimensional, by polishing or replacing dynamic obstacle avoidance forces, best direction force and related escape maneuver which are far from optimal but functional, and finally, by using any-angle algorithms in the pathfinding planning stage, such as $\theta*$ (Daniel et al. 2010) or incremental $\phi*$ (Nash, Koenig, and Likhachev 2009), or another post-smoothing algorithm coupled with $A*$. Swarm behavior and $A*$ is also an area where improvements can be made, by for example designing a "single file" procedure or a modified $A*$ that accounts for the swarm's size. The semi-stochastic exploration algorithm could also be "cleverer" by either integrating it with a dynamic, cognitive component (for example, by having drones explore some areas more intensively if they trigger more rescue interventions from these) or a pre-planned statistical component (by giving *ab initio* different weights to different areas, to influence semi-stochastic search), or both.

Operator querying could also be clearly improved, as the swarm-operator relationship is semantically minimal. Ideally, operators should be able to start a dialogue if necessary with the swarm, to further define their objectives or inform the swarm of special cases. This is achievable using NLP tools and through further research.

Regarding the simulation framework, extra precision (by, for example, reducing the acceptable error made by dynamic thrust computations with differential equation solving techniques) and the addition of atmospheric conditions to account for lower visibilities or wind components would be interesting upgrades; and finally, unexpected events such as drone sensing failures should also be testable and possibly make another use case for cognition.

Finally, on a more general level, the proposed system could be straightforwardly adapted to maritime/coastal operations and rescuing by 1) modifying existing behaviors or complementing them with new ones, and by 2) tailoring algorithms and environmental parameters of the simulation. Semi-stochastic search, for example, is not very efficient when obstacles are few, and could be improved with a measure of statistical bias: future positions of entities adrift at sea can be well anticipated using wind or sea current components. Adding these to the simulation would be an appropriate first step in tailoring the system to maritime environments.

Bibliography

Adams, Annette L. et al. (June 2007). "Search Is a Time-Critical Event: When Search and Rescue Missions May Become Futile". In: *Wilderness & Environmental Medicine* 18.2, pp. 95–101. ISSN: 10806032. DOI: 10.1580/06-WEME-OR-035R1.1. URL: https://linkinghub.elsevier.com/retrieve/pii/S1080603207702181 (visited on 05/15/2020).

Alfeo, Antonio L., Mario G.C.A. Cimino, and Gigliola Vaglini (Oct. 2019). "Enhancing Biologically Inspired Swarm Behavior: Metaheuristics to Foster the Optimization of UAVs Coordination in Target Search". In: *Computers & Operations Research* 110, pp. 34–47. ISSN: 03050548. DOI: 10.1016/j.cor.2019.05.021. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054819301340 (visited on 05/26/2020).

Atkinson, Richard C. and Richard M. Shiffrin (1968). "Human Memory: A Proposed System and Its Control Processes". In:

Baddeley, Alan D. and Graham Hitch (1974). "Working Memory". In: *Psychology of Learning and Motivation*. Vol. 8. Elsevier, pp. 47–89.

Berdahl, Andrew et al. (Feb. 1, 2013). "Emergent Sensing of Complex Environments by Mobile Animal Groups". In: *Science* 339.6119, pp. 574–576. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1225883. URL: https://www.sciencemag.org/lookup/doi/10.1126/science.1225883 (visited on 05/26/2020).

Boids (Flocks, Herds, and Schools: A Distributed Behavioral Model) (2020). URL: http://www.red3d.com/cwr/boids/ (visited on 02/07/2020).

Borenstein, J. and Y. Koren (June 1991). "The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots". In: *IEEE Transactions on Robotics and Automation* 7.3, pp. 278–288. ISSN: 1042296X. DOI: 10.1109/70.88137. URL: http://ieeexplore.ieee.org/document/88137/ (visited on 10/30/2019).

Borenstein, Johann and Y. Koren (1988). "High-Speed Obstacle Avoidance for Mobile Robots". In: *Proceedings of the 3rd International Symposium on Intelligent Control*, pp. 382–384.

Brandão, Alexandre Santos, Mário Sarcinelli-Filho, and Ricardo Carelli (2013). "An Analytical Approach to Avoid Obstacles in Mobile Robot Navigation". In: *International Journal of Advanced Robotic Systems* 10.6, p. 278.

Breiman, L. (1984). "Algorithm Cart". In: *Classification and Regression Trees. California Wadsworth International Group, Belmont, California.*

Bresenham, Jack E. (1965). "Algorithm for Computer Control of a Digital Plotter". In: *IBM Systems journal* 4.1, pp. 25–30.

Broadbent, D. E. (1958). *Perception and Communication. Elmsford, NY, US.* Pergamon Press. http://dx. doi. org/10.1037/10037-000.

Cherry, E. Colin (1953). "Some Experiments on the Recognition of Speech, with One and with Two Ears". In: *The Journal of the acoustical society of America* 25.5, pp. 975–979.

Chung, Soon-Jo et al. (Aug. 2018). "A Survey on Aerial Swarm Robotics". In: *IEEE Transactions on Robotics* 34.4, pp. 837–855. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2018.2857475. URL: https://ieeexplore.ieee.org/document/8424838/ (visited on 05/26/2020).

Coch, Donna, Lisa D. Sanders, and Helen J. Neville (2005). "An Event-Related Potential Study of Selective Auditory Attention in Children and Adults". In: *Journal of cognitive neuroscience* 17.4, pp. 605–622.

Cognition | Definition of Cognition by Lexico (2020). URL: https://www.lexico.com/en/definition/cognition (visited on 05/07/2020).

Communications, Tait (2012). *Race against Time - Emergency Response - Preventing Escalating Chaos in a Disaster.* URL: https://www.taitradio.com/__data/assets/pdf_file/0008/156077/Tait_WP_Race-Against-Time_US_v1_WEB.pdf (visited on 05/15/2020).

Daniel, K. et al. (Oct. 29, 2010). "Theta*: Any-Angle Path Planning on Grids". In: *Journal of Artificial Intelligence Research* 39, pp. 533–579. ISSN: 1076-9757. DOI: 10.1613/jair.2994. URL: https://jair.org/index.php/jair/article/view/10676 (visited on 03/08/2020).

Deutsch, J. Anthony and Diana Deutsch (1963). "Attention: Some Theoretical Considerations." In: *Psychological review* 70.1, p. 80.

Dorigo, Marco, Mauro Birattari, and Thomas Stützle (2006). "Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique". In: *IEEE Comput. Intell. Mag* 1, pp. 28–39.

Drag (Physics) (Apr. 23, 2020). In: *Wikipedia.* URL: https://en.wikipedia.org/w/index.php?title=Drag_(physics)&oldid=952685266 (visited on 04/30/2020).

Erdelj, Milan et al. (2017). "Help from the Sky: Leveraging UAVs for Disaster Management". In: *IEEE Pervasive Computing* 16.1, pp. 24–32.

Ferreira, André et al. (2008). "An Approach to Avoid Obstacles in Mobile Robot Navigation: The Tangential Escape". In: *Sba: Controle & Automação Sociedade Brasileira de Automatica* 19.4, pp. 395–405.

FitzGerald, Peter and Donald E. Broadbent (1985). "Memory for Attended and Unattended Visual Stimuli". In: *The Quarterly Journal of Experimental Psychology* 37.3, pp. 339–365.

Goodwin, Donald W. et al. (1969). "Alcohol and Recall: State-Dependent Effects in Man". In: *Science* 163.3873, pp. 1358–1360.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Hartman, Christopher and Bedrich Benes (July 2006). "Autonomous Boids". In: *Computer Animation and Virtual Worlds* 17.3-4, pp. 199–206. ISSN: 1546-4261, 1546-427X. DOI: 10.1002/cav.123. URL: http://doi.wiley.com/10.1002/cav.123 (visited on 10/01/2019).

He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Hocraffer, Amy and Chang S. Nam (Jan. 1, 2017). "A Meta-Analysis of Human-System Interfaces in Unmanned Aerial Vehicle (UAV) Swarm Management". In: *Applied Ergonomics* 58, pp. 66–80. ISSN: 0003-6870. DOI: 10.1016/j.apergo.2016.05.011. URL: http://www.sciencedirect.com/science/article/pii/S0003687016300989 (visited on 05/16/2020).

jburgoyn (Jan. 6, 2014). *Dynamic Thrust*. URL: https://itsallrc.wordpress.com/2014/01/06/dynamic-thrust/ (visited on 10/16/2019).

Jonker, Tanya R., Paul Seli, and Colin M. MacLeod (2012). "Less We Forget: Retrieval Cues and Release from Retrieval-Induced Forgetting". In: *Memory & Cognition* 40.8, pp. 1236–1245.

JSON (Apr. 27, 2020). In: *Wikipedia*. URL: https://en.wikipedia.org/w/index.php?title=JSON&oldid=953472069 (visited on 04/30/2020).

Kahneman, Daniel (1973). *Attention and Effort*. Vol. 1063. Citeseer.

Kahneman, Daniel, Rachel Ben-Ishai, and Michael Lotan (1973). "Relation of a Test of Attention to Road Accidents." In: *Journal of Applied Psychology* 58.1, p. 113.

Kakas, Antonis and Loizos Michael (2016). "Cognitive Systems: Argument and Cognition". In: *Cognitive Systems*, p. 7.

Kass, Gordon V. (1980). "An Exploratory Technique for Investigating Large Quantities of Categorical Data". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 29.2, pp. 119–127.

Khatib, Oussama (1986). "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots". In: *Autonomous Robot Vehicles*. Springer, pp. 396–404.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "Imagenet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*, pp. 1097–1105.

Leca, D. et al. (June 2019). "Sensor-Based Obstacles Avoidance Using Spiral Controllers for an Aircraft Maintenance Inspection Robot". In: *2019 18th European Control Conference (ECC)*. 2019 18th European Control Conference (ECC). Naples, Italy: IEEE, pp. 2083–2089. ISBN: 978-3-907144-00-8. DOI: 10.23919/ECC.2019.8795882. URL: https://ieeexplore.ieee.org/document/8795882/ (visited on 06/01/2020).

Levesque, Hector and Gerhard Lakemeyer (2008). "Cognitive Robotics". In: *Foundations of artificial intelligence* 3, pp. 869–886.

Lidar (Apr. 25, 2020). In: *Wikipedia*. URL: https://en.wikipedia.org/w/index.php?title=Lidar&oldid=953045738 (visited on 04/28/2020).

Lim, Sejoon and Daniela Rus (May 2012). "Stochastic Distributed Multi-Agent Planning and Applications to Traffic". In: *2012 IEEE International Conference on Robotics and Automation*. 2012 IEEE International Conference on Robotics and Automation, pp. 2873–2879. DOI: 10.1109/ICRA.2012.6224710.

Murphy, Robin R. (2014). *Disaster Robotics*. MIT press.

Nash, Alex, Sven Koenig, and Maxim Likhachev (2009). "Incremental Phi*: Incremental Any-Angle Path Planning on Grids". In: p. 7.

Newton's Laws of Motion (Mar. 8, 2020). In: *Wikipedia*. URL: https://en.wikipedia.org/w/index.php?title=Newton%27s_laws_of_motion&oldid=944492729 (visited on 04/30/2020).

O'Keefe, J. and J. Dostrovsky (Nov. 12, 1971). "The Hippocampus as a Spatial Map. Preliminary Evidence from Unit Activity in the Freely-Moving Rat". In: *Brain Research* 34.1, pp. 171–175. ISSN: 0006-8993. DOI: 10.1016/0006-8993(71)90358-1. URL: http://www.sciencedirect.com/science/article/pii/0006899371903581 (visited on 05/10/2020).

Pelánek, Radek et al. (2005). "Enhancing Random Walk State Space Exploration". In: *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems*, pp. 98–105.

Project Valhalla (Dec. 13, 2019). *Project Valhalla: Fast and Furious Java*. URL: https://www.javaadvent.com/2019/12/project-valhalla-fast-and-furious-java.html (visited on 04/26/2020).

Quinlan, J. Ross (1986). "Induction of Decision Trees". In: *Machine learning* 1.1, pp. 81–106.

Quinlan, J. Ross (1993). "C4. 5: Programming for Machine Learning". In: *Morgan Kauffmann* 38, p. 48.

Reid, Donald B. (1979). "An Algorithm for Tracking Multiple Targets". In: *IEEE Transactions on Automatic Control* 24, pp. 843–854.

Remy, Guillaume et al. (2013). "SAR.Drones: Drones for Advanced Search and Rescue Missions". In: p. 3.

Reynolds, Craig W (1987). "(~) ~ ComputerGraphics, Volume21, Number4, July 1987". In: p. 11.

Ruetten, Laik et al. (Jan. 2020). "Area-Optimized UAV Swarm Network for Search and Rescue Operations". In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). Las Vegas, NV, USA: IEEE, pp. 0613–0618. ISBN: 978-1-72813-783-4. DOI: 10.1109/CCWC47524.2020.9031197. URL: https://ieeexplore.ieee.org/document/9031197/ (visited on 06/14/2020).

Sampedro, Carlos et al. (June 2016). "A Flexible and Dynamic Mission Planning Architecture for UAV Swarm Coordination". In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2016 International Conference on Unmanned Aircraft Systems (ICUAS). Arlington, VA, USA: IEEE, pp. 355–363. ISBN: 978-1-4673-9334-8. DOI: 10.1109/ICUAS.2016.7502669. URL: http://ieeexplore.ieee.org/document/7502669/ (visited on 06/14/2020).

Scherer, Jürgen et al. (2015). "An Autonomous Multi-UAV System for Search and Rescue". In: *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use - DroNet '15*. The First Workshop. Florence, Italy: ACM Press, pp. 33–38. ISBN: 978-1-4503-3501-0. DOI: 10.1145/2750675.2750683. URL: http://dl.acm.org/citation.cfm?doid=2750675.2750683 (visited on 05/15/2020).

Senanayake, Madhubhashi et al. (Jan. 2016). "Search and Tracking Algorithms for Swarms of Robots: A Survey". In: *Robotics and Autonomous Systems* 75, pp. 422–434. ISSN: 09218890. DOI: 10.1016/j.robot.2015.08.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889015001876 (visited on 05/26/2020).

Shakeri, Reza et al. (Oct. 23, 2018). "Design Challenges of Multi-UAV Systems in Cyber-Physical Applications: A Comprehensive Survey, and Future Directions". In: arXiv: 1810.09729 [cs]. URL: http://arxiv.org/abs/1810.09729 (visited on 05/26/2020).

Simons, Daniel J. and Christopher F. Chabris (1999). "Gorillas in Our Midst: Sustained Inattentional Blindness for Dynamic Events". In: *perception* 28.9, pp. 1059–1074.

Staples, Gabriel (2019). *Propeller Static & Dynamic Thrust Calculation - Part 1 of 2*. URL: https://www.electricrcaircraftguy.com/2013/09/propeller-static-dynamic-thrust-equation.html (visited on 10/13/2019).

State of Valhalla (2020). URL: http://cr.openjdk.java.net/~briangoetz/valhalla/sov/01-background.html (visited on 04/26/2020).

Statheropoulos, M. et al. (2015). "Factors That Affect Rescue Time in Urban Search and Rescue (USAR) Operations". In: *Natural Hazards* 75.1, pp. 57–69.

Szegedy, Christian et al. (2015). "Going Deeper with Convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.

Treisman, Anne (1964). "Monitoring and Storage of Irrelevant Messages in Selective Attention". In: *Journal of Verbal Learning and Verbal Behavior* 3.6, pp. 449–459.

Ulrich, I. and J. Borenstein (1998). "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. IEEE International Conference on Robotics and Automation. Vol. 2. Leuven, Belgium: IEEE, pp. 1572–1577. ISBN: 978-0-7803-4300-9. DOI: 10.1109/ROBOT.1998.677362. URL: http://ieeexplore.ieee.org/document/677362/ (visited on 11/01/2019).

Ulrich, Iwan and Johann Borenstein (2000). "VFH/Sup*: Local Obstacle Avoidance with Look-Ahead Verification". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 3. IEEE, pp. 2505–2511.

Waharte, Sonia and Niki Trigoni (Sept. 2010). "Supporting Search and Rescue Operations with UAVs". In: *2010 International Conference on Emerging Security Technologies*. 2010 International Conference on Emerging Security Technologies (EST). Canterbury, TBD, United Kingdom: IEEE, pp. 142–147. ISBN: 978-1-4244-7845-3. DOI: 10.1109/EST.2010.31. URL: http://ieeexplore.ieee.org/document/5600072/ (visited on 05/15/2020).

Wolford, George and Fred Morrison (1980). "Processing of Unattended Visual Information". In: *Memory & Cognition* 8.6, pp. 521–527.

Appendices

Appendix A

Technical choices

A.1 Why Java?

There were two main options for developing this project: C++ or Java. Both languages are very evolved and are tailored for complex object-oriented programming projects - in fact, due to the nature of artificial intelligence algorithms, an abstraction-capable language (i.e. enabling concept abstractions via polymorphism, inheritance, etc.) is more than welcome; however they differ on one fundamental aspect: C++ code is compiled to native code whereas Java code is compiled into bytecode that can be executed by a virtual machine running on a chosen target environment.

In other words: C++ compiled code is more machine-specific but faster, whereas Java compiled code is less machine-specific but can end up being slower due to

- memory management: with C++ the developer has to manage memory by him/herself, with Java a garbage collector runs asynchronously and cleans unused objets from memory but this can lead to unwanted delays during execution
- virtual machine overhead and bytecode interpretation, although modern virtual machines compile parts of bytecode into native code (as is the case for Oracle's HotSpot[1] JVM[2] for example)

The first tested option was C/C++ and a graphic library based on OpenGL[3] or Vulkan[4] (NanoGUI[5] and ImGui[6]), with a multi-platform window management library (GLFW[7]). With C/C++ code compiled into native code, speed results were fantastic; however, trying to create a codebase compatible with Windows, Linux and MacOS simultaneously turned out to be too troublesome and time-consuming in the context

[1] https://openjdk.java.net/groups/hotspot/
[2] Java Virtual Machine
[3] https://www.opengl.org
[4] https://www.khronos.org/vulkan/
[5] https://github.com/wjakob/nanogui
[6] https://github.com/ocornut/imgui
[7] https://www.glfw.org

of a time-limited Master's dissertation implementation. Things would certainly be analyzed differently in the case of commercial video game development.

The second tested option was therefore Java. Initially, due to the high amount of calculations inherent to a computer simulation, a considerable amount of objects needed to be instantiated and dismissed and therefore the garbage collector was doing too much work resulting in slowdowns. A pooling strategy was implemented: it produced good results but increased complexity notably while cluttering the code with numerous extra instructions to take objects from/release objects to the pool; and overall it did not look completely adequate for the scope of this project. It appeared like tradeoffs would have to be made.

The good news, however, is that developers at OpenJDK[8] have been working for a few years now on a project called Valhalla. It is a modification of Java virtual machine specifications that enables the use of C++ struct-like data structures in Java. These are extremely interesting for computer-intensive projects as their storage in memory is optimized: they take less space and are much easier to reuse for memory management systems - hence enable considerable performance increases in certain use cases, such as computations involving a great number of memory accesses (*Project Valhalla* 2019), like scientific simulations. Although an experimental project to this date, project Valhalla will be merged with the mainstream JVM[9] tree in the years to come (*State of Valhalla* 2020), and is therefore future-proof.

[8] https://openjdk.java.net
[9] Java Virtual Machine

Appendix B

Algorithms

B.1 Distance from point to segment

Algorithm 5 Distance from point to segment

function SegmentToPointDistance(mental model M)
  $\sigma \leftarrow distance(A, B)^2$
  if $\sigma = 0$ **then return** $distance(A, P)$
  else
    $\tau \leftarrow \max(0, \min(1, \frac{\overrightarrow{AP} \cdot \overrightarrow{AB}}{\sigma}))$
    $P_{projected} \leftarrow translate(A, \tau\overrightarrow{AB})$
    return $distance(P, P_{projected})$
  end if
end function

B.2 A*

Algorithm 6 $A*$ algorithm

```
function ReconstructPath(node N)
    R ← new list
    R ← add N
    while preceding(N) exists do
        N ← preceding(N)
        R ← add N as first element
    end while
    return R
end function

function Heuristic(node N, node G)                    ▷ G is the goal node
    return distance(N, G)
end function

function F-Score(node N, node G)                      ▷ G is the goal node
    return g-score(N) + Heuristic(N, G)
end function

function AlphaStar(node S, node G)                    ▷ S and G are the start and goal nodes
    g-score(S) ← 0
    Set O ← new set
    O ← add S
    while ¬ empty(O) do
        Node C ← node with lowest F-Score in O
        if C = G then
            return ReconstructPath(C)
        end if
        remove first(O)
        for each Node N ∈ neighbors(C) do
            T ← g-score(C) + distance(C, N)
            if T < g-score(N) then
                preceding(N) ← C
                g-score(N) ← T
                O ← N
            end if
        end for each
    end while
    return empty list
end function
```

B.3 Bresenham algorithm

Algorithm 7 Bresenham algorithm

```
function BRESENHAM(cell A, cell B)
    list R ← new list
    x_0 ← x(A)
    y_0 ← y(A)
    x_1 ← x(B)
    y_1 ← y(B)

    d_x ← |x_1 − x_0|
    s_x ← x_0 < x_1 ? 1 : −1
    d_y ← −|y_1 − y_0|
    s_y ← y_0 < y_1 ? 1 : −1
    err ← d_x + d_y

    while True do
        R ← add cell(x_0, y_0)
        if x_0 = x_1 and y_0 = y_1 then
            return R
        end if
        e_2 = 2 err
        if e_2 ≥ d_y then
            err ← err + d_y
            x_0 ← x_0 + s_x
        end if
        if e_2 ≤ d_x then
            err ← err + d_x
            y_0 ← y_0 + s_y
        end if
    end while
    return R
end function
```