remote sensing

MDPI

Article

Energy-Efficient Navigation of an Autonomous Swarm with Adaptive Consciousness

Jawad Naveed Yasin [1,*], Huma Mahboob [2], Mohammad-Hashem Haghbayan [1], Muhammad Mehboob Yasin [3] and Juha Plosila [1]

[1] Autonomous Systems Laboratory, Department of Future Technologies, University of Turku, Vesilinnantie 5, 20500 Turku, Finland; mohhag@utu.fi (M.-H.H.); juplos@utu.fi (J.P.)
[2] Connected Shopping Ltd., Thetford IP24 1HP, UK; huma.mahboob@coursemerchant.com
[3] Department of Computer Networks, College of Computer Sciences & Information Technology, King Faisal University, Hofuf 31982, Saudi Arabia; mmyasin@kfu.edu.sa
* Correspondence: janaya@utu.fi

Abstract: The focus of this work is to analyze the behavior of an autonomous swarm, in which only the leader or a dedicated set of agents can take intelligent decisions with other agents just reacting to the information that is received by those dedicated agents, when the swarm comes across stationary or dynamic obstacles. An energy-aware information management algorithm is proposed to avoid over-sensation in order to optimize the sensing energy based on the amount of information obtained from the environment. The information that is needed from each agent is determined by the swarm's self-awareness in the space domain, i.e., its self-localization characteristics. A swarm of drones as a multi-agent system is considered to be a distributed wireless sensor network that is able to share information inside the swarm and make decisions accordingly. The proposed algorithm reduces the power that is consumed by individual agents due to the use of ranging sensors for observing the environment for safe navigation. This is because only the leader or a dedicated set of agents will turn on their sensors and observe the environment, whereas other agents in the swarm will only be listening to their leader's translated coordinates and the whereabouts of any detected obstacles w.r.t. the leader. Instead of systematically turning on the sensors to avoid potential collisions with moving obstacles, the follower agents themselves decide on when to turn on their sensors, resulting in further reduction of overall power consumption of the whole swarm. The simulation results show that the swarm maintains the desired formation and efficiently avoids collisions with encountered obstacles, based on the cross-referencing feedback between the swarm agents.

Keywords: autonomous swarm; multi-agent systems; energy efficient; swarm intelligence; leader follower; collision avoidance

Citation: Yasin, J.N.; Mahboob, H.; Haghbayan, M.-H.; Yasin, M.M.; Plosila, J. Energy-Efficient Navigation of an Autonomous Swarm with Adaptive Consciousness. *Remote Sens.* **2021**, *13*, 1059. https://doi.org/10.3390/rs13061059

Academic Editors: Józef Lisowski, Kouzou Abdellah, Haitham Abu-Rub, Piotr Szymak and Andrzej Stateczny

Received: 21 January 2021
Accepted: 5 March 2021
Published: 11 March 2021

1. Introduction

The optimization of autonomous navigation, collision avoidance, and resource allocation in swarms of drones (Unmanned Aerial Vehicles or UAVs) is currently one of the major focus areas in the robotics research community [1]. Besides the usability of individual UAVs, the considerable advantages of utilizing swarms of UAVs have increased their demand in various fields, such as search and rescue, traffic monitoring, atmospheric research, and military applications [2–5].

A general categorization of the agents in a swarm can be presented, as follows [6–8]:

- Evolutionary agents, where the agents work on the fundamental theory of evolutionary algorithms i.e., mutation, reproduction, recombination, and selection.
- Cognitive agents, where the agents can take decisions, make predictions, and process data based on their cognitive architecture.
- Reactive agents, where, as the name suggests, the agents react to a signal from another agent or to any change in the surrounding environment.

- Flocking agents, where the agents imitate the flocking behavior, i.e., moving together, inspired by flocks of birds or swarms of bees, for instance.

When considering the navigation of a swarm of drones, collision avoidance and formation maintenance in the swarm are the two of the most prominent problems to solve [9–11]. With the exponential increase in the use of UAVs and their integration into many different commercial, military, and leisure applications, the need for an efficient onboard collision avoidance system increases exponentially. Such an onboard system enables a drone to react rapidly to encountered objects during flight [12,13]. Owing to, e.g., onboard payload limitations, power limitations, and complications in remote monitoring, tasks become increasingly difficult for UAVs to accomplish. The robotics community is trying hard to counter these issues by developing new technologies to ensure safe navigation for UAVs in various environments [14–17].

Collision avoidance systems or algorithms are responsible for safely and reliably avoiding any possible collisions amongst the agents (e.g., UAVs) themselves and between an agent and a surrounding obstacle in the environment [18]. Collision avoidance algorithms can be roughly classified into the following three categories [19]: (1) sense-and-avoid algorithms that simplify the process by delegating the detection and avoidance activities to individual agents/nodes and, therefore, they have short response times, are independent of inter-node communication, and require less computational power [20–22]; (2) force-field algorithms, also known as potential field methods, which use the basic concept of attraction/repulsion between the agents in the swarm and between the agents and obstacles in the environment to guide an agent towards a destination while avoiding objects along the path [23–26]; (3) optimization based algorithms, relying on geographical information, which utilize knowledge on the sizes, shapes, and locations of obstacles to provide near-optimal path planning solutions [27–29].

Formation control can be divided into several separate tasks: navigation of the whole formation/swarm from one point to the designated point, maintaining a certain formation shape or orientation, splitting the formation, bringing the agents back into the original formation, and avoiding collisions while accomplishing these tasks [30]. In more general terms, a formation can be defined as the shape where the positioning of each agent within the swarm is relative to other agents [31–33]. Formation maintenance algorithms can be outlined into the following three classes [34,35]: (1) leader–follower based approaches, in which all of the agents in the swarm follow the leader and autonomously maintain their respective positions, w.r.t. their neighbours and the leader [36–39]; (2) virtual structure based approaches, in which all of the agents of the swarm as a whole are considered to be a single compound agent to be navigated along a given trajectory [40–43]; and, (3) behavior based approaches, in which the agents select their behavior in each situation based on a pre-determined procedure or strategy [44,45].

In this paper, we propose a strategy to reduce the processing power of individual agents in the swarm without losing the swarm's ability for autonomous operation. In order to achieve this, a leader-follower based approach is adopted for maintaining the formation, due to its relatively simple implementation, scalable nature, and reliability [9,38]. The global leader in the formation utilizes a given global collision avoidance algorithm, which is then used by the followers for the calculation of the relative coordinates, which are also known as translational coordinates, w.r.t. themselves, of the detected objects in the environment, as observed by the global leader. Furthermore, followers themselves, upon receiving the observed coordinates of a detected obstacle from the leader, compute to decide the optimal time for turning on their sensors for dynamic avoidance if some error has been detected in the calculation of translational coordinates while cross-checking is performed by the leader. In the case an error has been detected, it indicates that the obstacle is not stationary anymore, i.e., the environment is not static and, hence, it is treated as a moving obstacle, i.e., the environment is dynamic.

In this paper, a new strategy for reducing the overall energy consumption of a swarm is proposed. The main idea is to remove the unnecessary power consumption due to

the sensors for a portion of the swarm in cases where the information the agents perceive from their neighbors is enough for individual navigation of an agent. In other words, the neighbors of each agent are playing as a source of information for navigational purposes. The main contribution of the proposed approach is that besides the existing energy-efficient methods, such as movement-based and communication-based methods, the energy consumption of the swarm can be further reduced by injecting adaptive consciousness in the agents, especially in scenarios where the environment is either static or dynamic variables in the surroundings are negligible. In such situations, the agents can turn off their ranging sensors, translate the coordinates transmitted by their leader, and, upon performing necessary calculations, switch to the high-conscious mode if deemed to be necessary.

The organization of the rest of the paper is as follows. Section 2 provides the motivation. Section 3 provides the development of the proposed algorithm in detail. Section 4 provides the simulation results. Finally, Section 5 provides the concluding remarks, discussion, and future work.

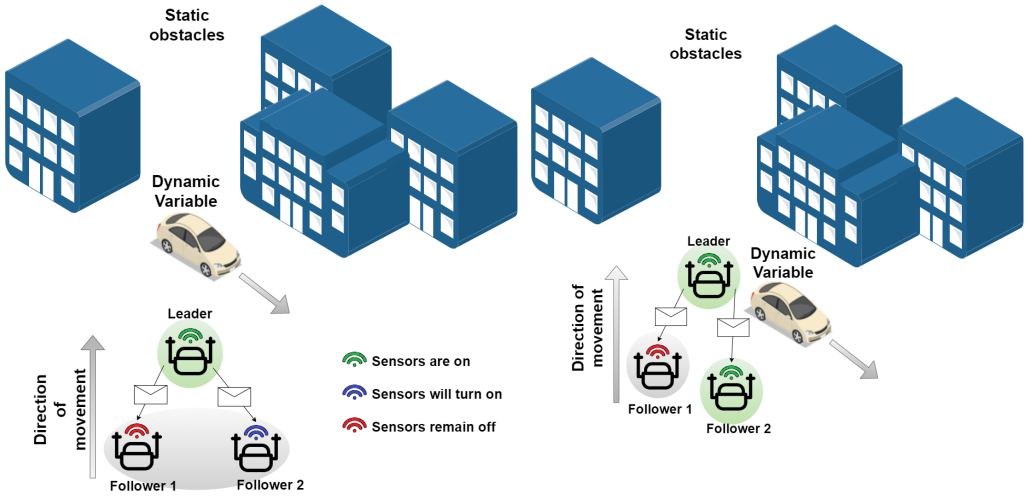
2. Motivation

The minimization of energy consumption of UAVs, or autonomous mobile agents more generally, for the efficient utilization of their limited resources, is currently gaining the attention of researchers. This is of high interest in scenarios where a prolonged mission duration is desirable, such as in large warehouses, underwater exploration, patrolling, and guarding [38]. However, in the literature, most of the energy-efficiency related work focuses on countering the adverse effect of external influences [46], route optimization [47], recharging optimization [48], altitude based navigation [49], or exploiting the direction of wind to extend mission duration [50].

In a multi-agent system, it is a norm to have every single agent making more or less intelligent decisions by utilizing the onboard sensor system for observing the surroundings. The active sensors and onboard processing of sensory data consume power and reduce battery run-time, which results in decreased mission life on a single charge. However, in static environments, all of the agents in the swarm do not necessarily have to either keep their onboard sensors turned on continuously or take decisions intelligently. If intelligent decision making and sensor usage are restricted based on some constraints, depending on the environment and surroundings, a major portion of power consumption due to sensors and related data processing can be reduced, as shown in Figure 1. The approach that is presented in [17] proposes a translational coordinates scheme in which the leader agent has its sensor turned on all the time, while the follower agents turn on their sensors as soon as the leader detects any static obstacle in its surroundings. Indeed, for such environments, if only the leader of the swarm remains at a high-conscious mode continuously and performs all of the necessary required calculations (Figure 1a), the system's sensor related power consumption can be reduced considerably, and the potential run-time of the mission is thereby increased. Furthermore, the adaptive capability of the individual follower agents to dynamically switch between the low-conscious and autonomous high-conscious modes, depending on the situation at hand (Figure 1b), guarantees that the system's degree of autonomy is not compromised.

In this paper, based on the principle that is described above, an approach is developed to reduce the overall power consumption of the system by intentionally deactivating and reactivating the sensors at run-time. Basing our proposed technique on reactive agents [6] and a control methodology for formation based on 1-N leader-follower approach (1-leader and N-followers) [38], the leader constantly observes the environment and informs its followers of the coordinates of any detected obstacles in the vicinity. In case there are no obstacles in the vicinity of the leader, it only broadcasts its own coordinates. The follower agents, after translating the coordinates according to their own positions, decide whether continuing the same trajectory is safe or if they are required to deviate from their path. However, in the case a dynamic obstacle is detected by the leader agent, the coordinates

are broadcasted in a similar manner to the followers, who, then upon performing the necessary calculations, realize the direction of movement of the obstacle and its speed. After this, based on their own trajectory and the obstacle's trajectory, it is calculated if continuing on the same trajectory will lead to a potential collision, in which case the respective agent switches from the low-conscious mode to the autonomous high-conscious mode, as shown in Figure 1. After successfully avoiding the collision or bypassing the obstacle, the follower agent switches back again to the low-conscious mode. Individual agents in the swarm utilize the collision avoidance technique developed and presented in [9] for avoiding collisions in the high-conscious mode. The main motivation behind this approach is not only the overall power cost reduction of the swarm, but also the capability of the agents to autonomously decide on switching between the low-conscious and high-conscious modes, enabling situation-aware optimization of power consumption, due to ranging sensors, at run-time. A simplified approach has been chosen for validation of the proposed approach by limiting the depth of the swarm to two agents, i.e., "N" = 2 agents in the 1-N leader–follower model. It is important to note here that the agents in the Follower_Mode are assumed to utilize the on-board IMU (Inertial Measurement Unit) and GPS (Global Positioning System) for localization purposes, i.e., obtaining their own positions (localization methodologies are not in the scope of this work. However, while in low-conscious mode, the possible error in obtaining position vectors due to the IMU drift over time and also the probable GPS signal loss, is also handled by the algorithm, as the cross-checking of the translated coordinates will return false readings and the follower agent will go into the high-conscious mode temporarily).



(**a**) Scene 1 (**b**) Scene 2

Figure 1. Swarm in a static environment with only one dynamic obstacle. (**a**) the initial scene where the leader (encircled by light green) has its ranging sensors turned on, performs all the processing, and transferring the information about the surroundings to its followers, (**b**) the Follower 2, upon performing necessary calculations, realizes that the dynamic obstacle is potentially moving in its path and turns on its sensors to be able to perform collision avoidance actively. In both scenes, Follower 1 does not turn its sensors on, since the dynamic obstacle is moving in the other direction and does not pose a potential collision threat. For bypassing the static obstacles, Follower 1 translates the received coordinates to generate escape routes.

3. Proposed Approach

In this section, we describe the proposed method in more detail. The strategy is to combine the navigation and object detection with coordinate calculation and adaptive autonomous modes in order to facilitate the process of autonomous swarm navigation using translational coordinates. The proposed top-level algorithm to accomplish this (Algorithm 1) is composed of two partial feedback-based algorithms: one for managing

the followers and the other for managing the adaptive autonomous mode in the presence of obstacles. The adaptive autonomous mode calls collision avoidance from within if deemed to be necessary.

In the Follower_Function module, drones receive the coordinates of their leader and of the obstacles, as seen and transmitted by the leader, and then perform the necessary calculations to translate the received coordinates according to their own respective positions. Meanwhile, if there is no feedback from the leader and the defined timeout is exceeded, it is assumed that the leader is not alive/reachable, i.e., the communication link with the leader is lost. In such a case, the drone in question temporarily declares itself as its own leader as a fail-safe mechanism and resumes navigation by turning its sensors on.

The environment is declared to be dynamic, if the leader detects one or more moving obstacles, or if the cross-check of the translational coordinates indicates a mismatch. The follower, in this case, starts calculating the obstacle's velocity to determine the apparent point of impact. Upon estimating the point of impact, the follower node/drone itself decides when it needs to turn on its sensor(s) in order to perform active collision avoidance for safe maneuvering.

Algorithm 1 Leader: Navigation & Object Detection

```
   agent Leader()
2:     while True do
           if obstacle detected then
4:             D_obstacle, A_obstacle ← Obstacle's distance and angle calculation;
           end if
6:         for all i do
               Follower(i).Transfer_Coordinates(d_tf, d_ff(i), t_success);
8:             if t_success then
                   cal.ref.coords ← Reverse cross-check follower's received
                                    coordinates(d_ff(i));
10:                cal.ref.angle ← Reverse cross-check follower's received angle(d_ff(i));
                   Δ_coords, Δ_angle = ref.coords − cal.ref.coords, ref.angle − cal.ref.angle;
12:                if |Δ_coords| > Threshold_c OR |Δ_angle| > Threshold_a then
                       Follower(i).Set-Dynamic;          ▷ Setting remote Dynamic flag
14:                end if
               end if
16:        end for
           Collision_Avoidance();
18:    end while
   end agent                                              ▷ end agent leader
```

3.1. Agent Leader

Algorithm 1 provides the general pseudo-code for the global leader. This top-level algorithm is executed utilizing the on-board processing units by all agents locally. The assignment of IDs to all agents is assumed to be achieved before starting the mission. The algorithm is initialized by first setting up the necessary flags. Subsequently, the leader agent starts scanning for any obstacles in the vicinity and then calculates the distance and angles at which the detected obstacle(s) lie (Lines 3–4). After this, the leader starts sending its coordinates to its respective follower(s), including the detected obstacle's (if any) angle and distance (Lines 6–7). Here, the symbols, d_{tf} and d_{ff}, are used to simplify the notations and contain the distance of the obstacle(s), the angles at they lie, the agents' own parameters, such as coordinates and heading direction. After receiving the coordinates from its follower(s), i.e., the transfer is successful ($t_{success}$), the leader cross-checks the distances and angles calculated by the follower (Lines 8–10). After cross-checking, the environment is declared to be dynamic if the absolute value of the received angles or coordinates, i.e., $\Delta_{coords}, \Delta_{angle}$, is more than a defined threshold level, i.e., $Threshold_c$

error tolerance for coordinates and $Threshold_a$ error tolerance for calculated angle (Lines 12–14). The leader agent then calls *Collision_Avoidance* to bypass the obstacle(s) successfully (Line 17).

3.2. Agent Follower

Algorithm 2 provides the general pseudo-code for the follower agents. The algorithm starts by declaring its global procedures that are also utilized by the leader and, furthermore, setting up the necessary *flags* (Lines 2–3). Afterwards, the follower agent(s) check if the leader is alive, i.e., they are getting constant feedback from the leader (*Lead_is_Alive* == *True*), or if they are in the follower mode, i.e., *Follower_Mode* == *True*. If either *Lead_is_Alive* or *Follower_Mode* is False, the respective agent performs obstacle detection actively and then calculates the distance and angle at which the obstacle is detected (Lines 5–6). In the case of constant feedback from the leader, the respective follower agents check if the environment has been declared as dynamic by the leader or not. If the environment is not declared as dynamic, the agent calls *Follower_Function* for navigational purposes. On the contrary, if the environment has been declared to be dynamic, *Adaptive_Mode_Function* (Lines 8–11). Finally, the *Coordinates_Received* flag is reset to False, for next iteration (Line 13).

Algorithm 2 Follower: Navigation & Object Detection

```
    agent Follower()
2:      global procedure Transfer_coordinates, Set-Dynamic;▷ declaration of of follower's
                                                    global procedures Leader will call
        Lead_is_Alive, Follow_Mode, Dynamic, coordinates_received = True, True, False,
                                                                        False;
4:      while True do
            if (!Lead_is_Alive OR !Follower_Mode) AND obstacle detected then
6:              D_obstacle, A_obstacle ← Obstacle's distance and angle calculation;
            end if
8:          if !Dynamic AND Lead_is_Alive then
                Follower_Function();
10:         else
                Adaptive_Mode_Function();
12:         end if
            coordinates_received = False;
14:     end while
    end agent                                                   ▷ end agent follower
```

3.3. Set-Dynamic

Algorithm 3 shows the procedure that is called by the leader to set the dynamic flag for its followers to True.

Algorithm 3 Set-Dynamic

```
    procedure SET-DYNAMIC() ▷ this procedure is called remotely to set Dynamic to True
                                                                    for Follower(i)
2:      Dynamic = True;
    end procedure
```

3.4. Transfer Function

Algorithm 4 specifies the pseudo-code for Transfer Function. The algorithm starts by setting the $t_{success}$, i.e., the transfer success, flag based on the *coordinates_received* and randomizing the True/False to simulate the possibility of transfer failure (Line 2). Subsequently, if $t_{success}$ is True, the data sent by the leader are utilized to compute the translated coordinates by the follower (d_{ff}). If there was a constant feedback from the leader, the node/follower uses the translational coordinates of the obstacle, as observed by the

leader and based on its own coordinates, calculates the location of the obstacle according to its own position, as shown in Figure 2 variables description given in Table 1). These translated coordinates that are calculated by the follower are then utilized by the leader, as data from the follower, d_{ff}, (Algorithm 1, Lines 17–19) for cross-checking purposes. The *coordinates_received* flag is then set to *True* (Lines 3–6).

Algorithm 4 Transfer Function

procedure TRANSFER_COORDINATES(d_{tf}, d_{ff})
2: $t_{success}$ = !*coordinates_received* AND random select(True, False); ▷ modeling transfer failure possibility
if $t_{success}$ **then**
4: *receiveddata* = d_{tf};
d_{ff} = calculate translational coordinates(d_{tf}); ▷ Follower's data for Leader
6: *coordinates_received* = True;
Lead_is_Alive = True; ▷ if *Lead_is_Alive* is False, turn it to True
8: **end if**
end procedure

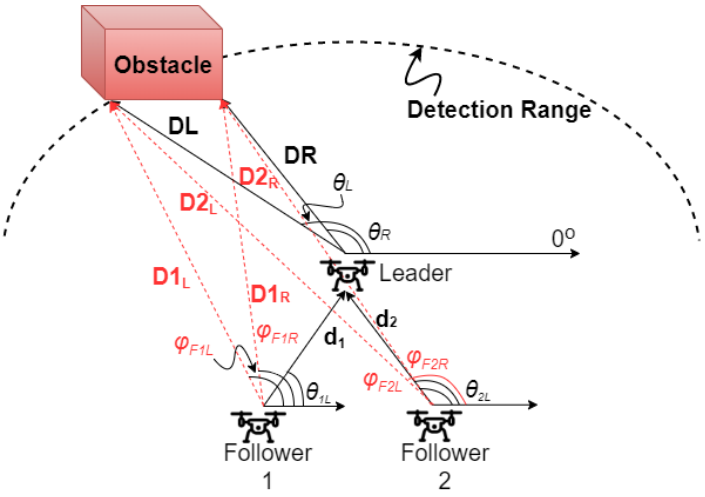


Figure 2. Distance and direction calculation of the detected obstacle.

Table 1. Description of Variables from Figure 2.

Variables	Description
DL DR	distance of the obstacle's left and right edges from leader
$D1_L$ $D1_R$ $D2_L$ $D2_R$	translational calculated distance of right and left edges of the obstacle from follower 1 and follower 2, respectively, as observed by leader
d_1 d_2	follower 1 and follower 2's distance from leader respectively
θ_R θ_L	angle of detected right and left edges from leader respectively
θ_{1L} θ_{2L}	leader's angle from follower 1 and follower 2, respectively
φ_{F1R} φ_{F1L} φ_{F2R} φ_{F2L}	angles of right and left edges as calculated by follower 1 and follower 2, respectively

3.5. Follower Function: Coordinate Calculation

Algorithm 5 specifies the *Follower_Function* module, where the follower node waits until either the defined *Timeout* is reached or the coordinates are received (Line 2). Subseuently, if the *coordinates_received* is True, i.e., coordinates are received by the follower, the reference coordinates that the follower should navigate to, i.e., *ref.coords*, are updated w.r.t. d_{ff} (Lines 3–4). Moreover, if the ranging sensors are on/activated previously, they are turned off, as the agent is now in the follower mode (lines 5–6). However, if the coordinates are not received until the *Timeout*, the *Lead_is_Alive* is set to *False*, ranging sensors are turned on for actively observing the environment and perform obstacle avoidance, and to make itself its own leader, the node sets the *ref.coords* as its own coordinates, i.e., *self.coords*, to start navigating towards the destination (Lines 8–11).

Algorithm 5 Follower Function

```
    procedure FOLLOWER_FUNCTION()
2:      wait until Timeout OR coordinates_received;
        if coordinates_received then
4:          ref.coords ← d_ff; ▷ Update the reference coordinates to translational coordinates
            if Sensors are ON then
6:              Turn off the Sensors;
            end if
8:      else
            Lead_is_Alive = False;
10:         Turn on the Sensors;
            ref.coords = self.coords;
12:     end if
    end procedure
```

The timeout signal is only checked if the node in question is not the leader itself. Every other node constantly checks whether its respective leader's transmitted signals are being constantly received. If the node has not received the coordinates sent by its leader by the timeout, it turns on its sensors for active collision avoidance maneuvering by declaring itself as its own leader.

3.6. Adaptive Mode Function

This module, Algorithm 6, is called by the leader if it detects obstacles in the vicinity. Or, in the case the environment is already as dynamic, then the nodes in *Follow_Mode* call this module locally. As soon as this module is called, it is checked whether the node in question is the global leader or is it one of the followers (Line 2). The default or initial value of *Follow_Mode* for all of the follower nodes is *True*. If the node is the follower, using previous translational readings, the obstacle's velocity is approximated (Line 3) using Equations (1)–(3), and a visual illustration is shown in Figure 3.

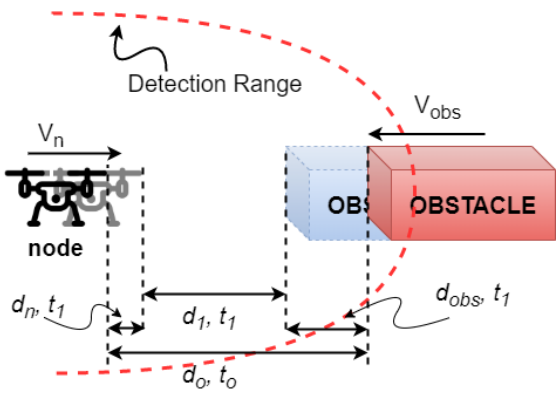


Figure 3. Illustration of calculation of movement of obstacle.

Algorithm 6 Adaptive Mode Function

```
procedure ADAPTIVE_MODE_FUNCTION()
    if Follow_Mode then
        V_obs ← Calculate obstacle's velocity from previous readings(d_tf);
        D_imp ← Calculate the distance to impact;
        if D_imp < Detection Range then
            Turn on the sensors;
            Follow_Mode = False;
        end if
    end if
    Collision avoidance ();
    if D_obstacle NOT in Detection Range then
        Dynamic = False;
        if Leader != Self AND Lead_is_Alive then
            Follow_Mode = True;
            Turn off the sensors;
        end if
    end if
end procedure
```

Three scenarios are possible based on V_{obs}: (1) if $V_{obs} < 0$, i.e., positive, this means that the obstacle is going away from the node; (2) if $V_{obs} = 0$, this means that the obstacle is not moving, i.e., stationary; (3) if $V_{obs} < 0$, this means that the obstacle is travelling towards the node. Based on these readings, the distance to the potential impact/collision (D_{imp}) is calculated (Line 4). The distance travelled by the node after t_1 seconds is computed by:

$$d_n = v_n * (t_1 - t_0) \tag{1}$$

where d_n and v_n are the distances travelled by the node and velocity of the node, respectively. The distance covered by the obstacle and its velocity can be determined by (2) and (3), respectively:

$$d_{obs} = d_o - d_n - d_1 \tag{2}$$

$$v_{obs} = d_{obs}/\Delta t \tag{3}$$

where d_{obs} is the distance travelled by the obstacle, v_{obs} velocity of the obstacle, d_n the distance travelled by the node, d_1 the distance between the obstacle and the node at time t_1, and d_o is the distance between the obstacle and node at the previous time t_0.

If the distance that is travelled by the obstacle is zero, i.e., $d_{obs} = 0$, it means that the obstacle is stationary. However, if the distance covered by the obstacle after t_1, i.e., d_1 is less than the distance between the node and the obstacle (d_o), in that case, the obstacle is moving towards the node (as shown in Figure 3). Otherwise, the obstacle is moving away from the node. Based on the movement of the node and the obstacle, the apparent distance to impact is calculated and, if that distance to impact is less than the detection range of the on-board sensor system, the node turns on its sensors and comes out of follower mode to perform active collision avoidance (Lines 5–7). The collision avoidance module is then invoked to constantly monitor the environment (line 10). After successful collision avoidance, the status of the surroundings is changed back to static (Lines 11–12). The node turns off its sensors and goes back into *Follow_Mode* and the control is returned to the main module (Lines 13–16).

3.7. Collision Avoidance

Collision avoidance, with the pseudo-code in Algorithm 7, is invoked when the detected obstacle gets critically close. It is then checked whether there is only one obstacle in the vicinity or there are multiple obstacles (Line 3). In case, more than one obstacle is detected, the gap between the obstacles is calculated (Line 4). Based on the calculation,

the algorithm takes the following actions: if the gap is more than the $safe\_dist$, i.e., defined based on the dimensions of the UAV, then the UAV is aligned to pass through the gap available between the obstacles (Lines 5–6); otherwise, the obstacles are treated as a single obstacle and the UAV is rerouted accordingly to navigate around them in order to avoid collisions (Lines 7–9). However, if only one obstacle was detected in the first place, the path planning is done accordingly to bypass the obstacle while keeping the deviation to a minimum (Lines 11–13). For performing successful path planning and aligning, we utilized and implemented the technique that is presented in [9]. The algorithm then recalculates the obstacle's distance and updates it. Based on the updated value, the control returned back to the Adaptive_Mode_Function.

Algorithm 7 Collision Avoidance

```
    procedure COLLISION_AVOIDANCE()
2:      while D_obstacle < Detection Range do
            if detected obstacles > 1 then
4:              gap ← calculate the gap between obstacles;
                if gap > safe_dist then
6:                  path planning(edges);    ▷ UAV is aligned to pass through the obstacles
                else
8:                  path_plan ← single obstacle;
                    path planning(path_plan);              ▷ considered as single obstacle
10:             end if
            else
12:             path_plan ← single obstacle;
                path planning(path_plan);
14:         end if
            D_obstacle ← Update the obstacle distance;
16:     end while
    end procedure
```

4. Simulation Results

The area used for the basis of the simulation was defined as 700 m × 500 m two-dimensional XY-plane, i.e., all of the objects are considered to be at the same altitude. The number of agents is set to three and the agents are already in the defined V-shaped formation at the start of the mission. It is important to note that, in the performed experiments, the leader agent has more computational and power resources to be able to perform the leadership tasks. The leader was equipped with Nvidia Jetson TX2, which is a power-efficient embedded system with the capability of operating between 0.5–2 GHz. In order to be more power efficient, we set the operating frequency to 0.5 GHz, at which the average power consumption is 4.5 Watts. The followers were equipped with Raspberry Pi 3B, which consumes around 2.4 Watts in high-conscious mode and 1.4 Watts in the idle mode. Velodyne Puck LITE was used for the generation of the data, which was then used in the simulation platform for visualization purposes, simulating the sensor, generating the obstacles, and subsequently to verify the proposed algorithm. Puck LITE has a 360° field of view horizontally, 30° vertically, generates 300,000 points/second, and has an accuracy of ±3 cm and range of 100 m [51]. For communication between the agents, due to its longer transmission range (100 m indoor/urban environment), data rate of 250 kbps, and possibility of large number of devices to be connected, communication-based power consumption is evaluated based on Legacy Digi XBee-Pro S1 802.15.4 consumption, i.e., in Transmitting Mode = 710 mW, in Receiving Mode = 182 mW [52].

The following assumptions and initial conditions are used in this study:

1. all of the agents are travelling with constant ground speeds;
2. communication link between the agents is setup ideally and with no information loss; and,
3. agents utilize the on-board localization techniques in order to obtain their positions.

The LiDAR (Light Detection and Ranging) data [53] is shown at different time intervals and from different angles shown in Figure 4. Figure 4b–d show the obstacles when they are in close vicinity and in the detection range. These figures show the position of the LiDAR sensor equipped drone as the blue, red, and green interactive marker. It is to be noted that the developed algorithm takes the point cloud that is captured by the LiDAR based on the defined constraints to reduce the complexity of the algorithm. Because its running on a resource-constrained system it is crucial to discard unnecessary point clouds, i.e., point clouds that are not in the field of view (FoV). The FoV is defined in the proposed algorithm to be $\pm 30°$. Only the obstacles within this FoV are considered to be posing a potential collision threat to the agents and, therefore, any obstacles outside this are discarded. The detected obstacles are contoured by light blue rectangles.

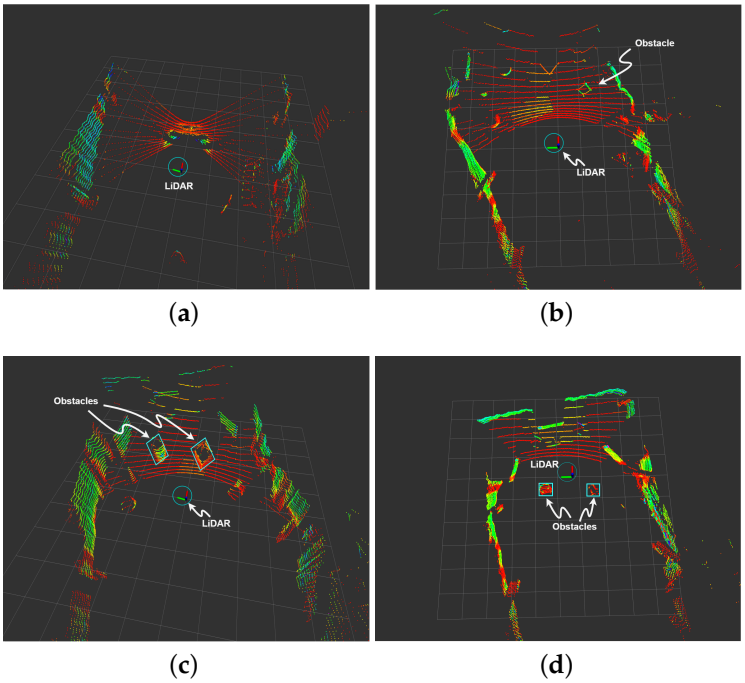


Figure 4. Light Detection and Ranging (LiDAR) point set snapshots at different intervals: starting point of simulation and when the obstacles are visible. (**a**) LiDAR data 1: at the start of the simulation. (**b**) LiDAR data 2: obstacles are starting to enter the detection range. (**c**) LiDAR data 3: obstacles are detected and in close proximity. (**d**) LiDAR data 4: bypassing the obstacles.

Figure 5 shows the simulation results for a static environment when the obstacles are stationary. In the V-shaped formation that is shown in the figure, agent 1 (leader) is shown as a blue circle, which has its ranging sensor (shown in Figure 4) on always-on mode, while red and green followers, i.e., agent 2 and agent 3 respectively, are following the leader by translating the leaders' transmitted coordinates as shown in Figure 2. In Figure 5b, the obstacle has entered the detection range of agent 1 but since there is no danger of collision, agent 1 continues its trajectory as shown in Figure 5c. Similarly, agent 2 is following agent 1 while utilizing the translational coordinates, and as there is no collision risk found upon calculations it continues to function in the *Follower_Function*. Whereas, as shown in Figure 5b, the translational coordinates calculations performed by agent 3 clearly indicate the potential collision if the same trajectory is continued and, therefore, it diverts from its original trajectory by performing offline collision avoidance to avoid the obstacle, as can be seen from the traces of the agents in Figure 5c.

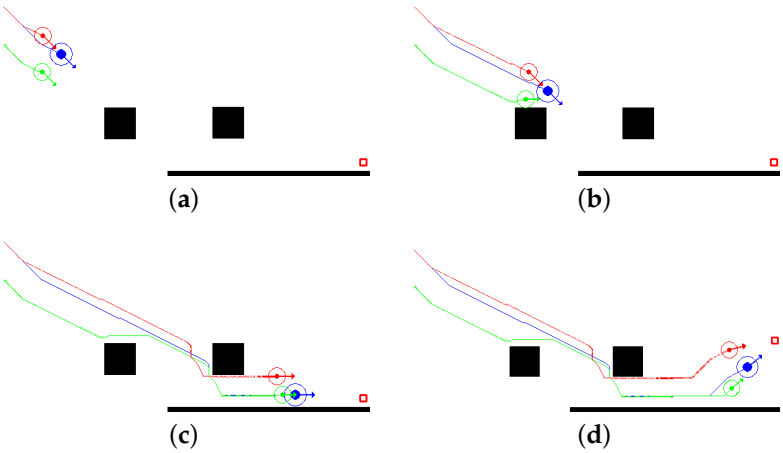


Figure 5. Simulation results at different time intervals, showing the behavior of the agents while going through obstacles. (**a**) Mission start with agents in formation, with leader in blue, agent 2 in red, and agent 3 in green. (**b**) Obstacle in range, agent 3 deviates by performing necessary calculations. (**c**) Traces of movement while going through the obstacles. (**d**) Agent 3 comes back to its intended position in the formation when there is no obstacle.

Agent 2 maintains its trajectory w.r.t. the leader without deviating from the original trajectory even while going through the obstacles and second obstacle being close to it, as can be seen from Figure 5c,d. This shows the effectiveness of utilizing the translational coordinates as the obstacle was out of the collision radius as also indicated by the performed calculations by the agent. Whereas, agent 3 maintains the pre-defined minimum distance from agent 1 after avoiding the obstacle, which is due to the third obstacle in the bottom. In Figure 5d, the destination point was moved at run-time to demonstrate the reformation process and, as can been seen, agent 3 starts going back to the position in the formation, it was supposed to be, as soon as it moves away from the obstacle in the bottom.

Figure 6 illustrates the simulation results for dynamic environment scenarios, where: *Case 1:* only one obstacle is moving, as shown in Figure 6a–c. In this case scenario, the obstacle is moving in the direction of the swarm. The obstacle, as observed by agent 1, will intercept the swarm and a potential collision is evident if agent 3 continues its trajectory, as can be seen from Figure 6b. Therefore, after performing the necessary calculations, agent 3 turns on its own ranging sensor and performs collision avoidance actively. After avoiding the obstacle, agent 3 turns off its sensors and starts following the leader once again based on translational coordinates from the leader. Figure 7a shows the distance maintained by the agents throughout the simulation; *Case 2:* two obstacles are moving, as shown in Figures 6d–f. In this case scenario, the first moving obstacle only disturbs agent 3, as in Case 1. However, after performing the translational coordinate calculations, as soon as agent 2 observes that the second obstacle is moving towards their direction and a collision may be inevitable, it turns on its ranging sensor to actively avoid the collision and divert from its trajectory if necessary (as shown in Figure 6f). After successful avoidance, when there are no moving obstacles in the vicinity, the agents turn off their sensors and switch back to the *Follower_Function*. Figure 7b shows the distance maintained by the agents throughout the simulation.

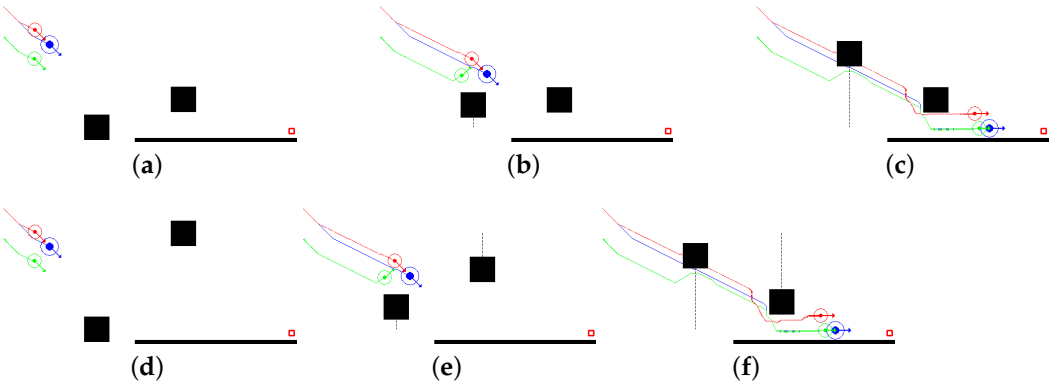
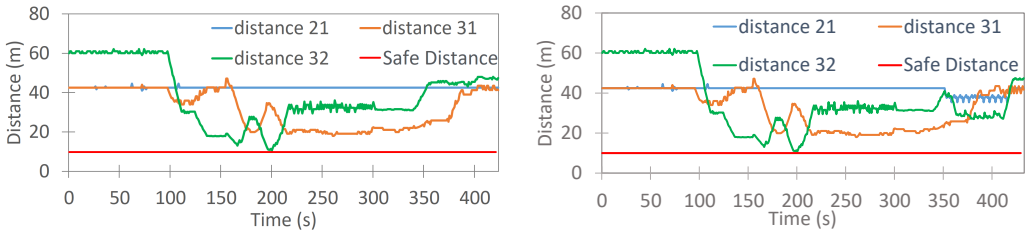


Figure 6. The results shown from the simulations of two case scenarios showing the traces of movement and deviations due to moving obstacles, with one moving obstacle and two moving obstacles. (**a**) Mission start. (**b**) Moving obstacle observed. (**c**) Overall trace of movement. (**d**) Mission start. (**e**) Movement of Obstacle 1 observed. (**f**) Movement obstacle 2 observed and traces of the deviated path.



(**a**) With one moving obstacle (**b**) With two moving obstacles

Figure 7. Distance maintained by each agent from its neighbour.

We report and compare the power consumed by our setup based on the experiment to analyze the efficiency of the proposed algorithm. The battery used in the setup for calculation of these attributes was 5000 mAh and, in typical conditions, the power consumption of one Velodyne Puck LITE is 8 Watts and operating voltage is 9 volts. Based on these values and the tracking the amount of time, each agent had its sensor turned on during the simulation with and without our proposed methodology, as shown in Figure 8. We placed these calculated values side-by-side with the reference algorithm [17]. The normal setup with all agents running their sensors always-on mode will consume about 3000 mWh, which is calculated as

$$\sum_{agent=1}^{N} E_{agent} = P_{s,agent} * t_{\alpha,agent} \tag{4}$$

where the summation is over all the agents in the system, E_{agent} denotes the energy consumption of an agent, $P_{s,agent}$ is the power consumption of the agent's sensor, and $t_{\alpha,agent}$ is the total time for which the agent's sensor remains active. In the instant case, there are three agents, the power consumption of the sensor is 8000 mW and the sensors remain active all the time, i.e., 450 s, E_{agent} comes out to be 1000 mWh and the total consumption for three agents will be 3000 mWh.

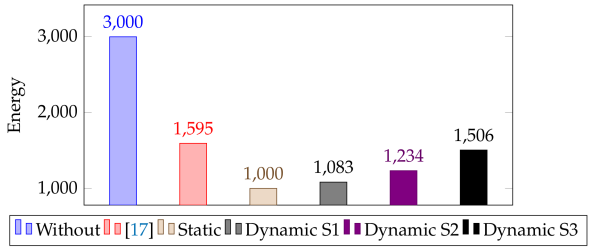


Figure 8. Energy Consumption by all nodes (mWh).

Similarly, the setup that was used in [17] consumed 1595 mWh, as in that setup the follower agents turn on the sensors as soon as the leader detects any static obstacle. However, while utilizing the proposed modified algorithm, the energy consumption for the same setup can be reduced by about 600 mWh to 1000 mWh. As in the proposed methodology, only agent 1 (the leader) had its sensor always-on, while agents 2 and 3 never had to turn on their sensors, as they only translated the broadcasted coordinates by the leader and followed. Subsequently, the power consumption due to ranging sensors in static environments can be further reduced by another 40% mark approximately. Additionally, that can be further utilized when the environment has some dynamic variables involved and, hence, increasing the overall mission life on a single charge. When testing the proposed methodology for the environment with some dynamic variables, the following results were observed: (1) Dynamic S1: is the experiment with swarm consisting of three agents and obstacle 1 moving. This only affected agent 3, and agent 3 turned on its sensor to perform collision avoidance for 37s and turned off the sensor after successful avoidance to go back into the *Follower_Function* resulting in the overall energy consumption due to sensor usage by all nodes to 1083 mWh; (2) Dynamic S2: is the experiment performed with swarm that consists of three agents and obstacle 1 and 2 are both moving. This affects both agents 2 and 3. Agent 3 had to turn on its sensor for 37 s, while agent 2 had its sensor turned on for 68 s, which resulted in the overall energy consumption due to sensor usage by all nodes to 1234 mWh; and, (3) Dynamic S3: is the experiment performed with swarm consisting of five agents and both obstacles 1 and 2 are dynamic. This affects all of the agents. Agents 2 and 3 remain at high-conscious mode for the same amount of time as in "Dynamic S2" case, whereas agents 5 and 4 remain at high-conscious mode for 57 s and 65 s, respectively. Resulting in the overall energy consumption due to sensors usage to 1506 mWh.

Based on the incremental consumption due to dynamic obstacles in the environment, Figure 9 shows an estimation of the relationship between the dynamicity of the environment and the energy consumption of the swarm. It is important to note that the energy consumption due to the sensor(s) usage depends on the duration that each agent stays at the high-conscious mode. It can be concluded that the energy consumption of the swarm is independent of the number of dynamic obstacles; however, it is dependent on the duration that each agent is affected by the dynamic obstacles or has to stay at high-conscious mode, i.e., their sensor(s) turned on. In the figure, $A = 1$ represents one agent operating in high-conscious mode, $A = 2$ represents two agents operating in the high-conscious mode, $A = 3$ shows the consumption of the swarm when three agents are operating in the high-conscious mode, and $A = 5$ shows the consumption of the swarm when five agents are operating in the high-conscious mode.

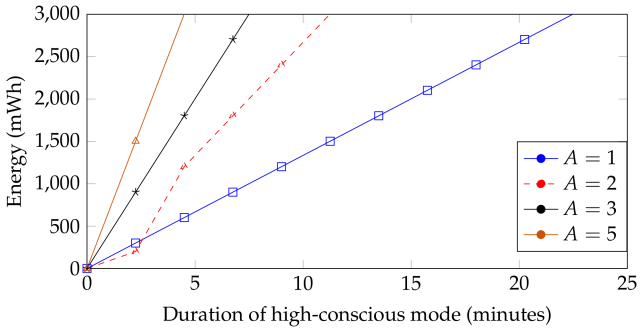


Figure 9. Energy consumption of the swarm based on the average duration that the agents stay at high-conscious mode.

5. Conclusions and Future Work

We developed an algorithm for a pre-defined formation of multi-agents with adaptive ability for static and dynamic environments to reduce the power consumption due to sensors usage. Under normal conditions in a static environment, one dedicated agent

(leader) can see its surroundings with the sensors turned on, while other agents (followers) blindly follow by translating the received coordinates from the leader. However, according to the demand of the environment, the follower(s) have the ability to decide to turn on their ranging sensors for safe maneuver, after performing the necessary calculations. The proposed methodology of the effectiveness of translational coordinates and the proof of concept was verified in the simulation environment. It is evident from the simulation results that the proposed method is reliable in static environments, as well as in the environment with some dynamic variables. Furthermore, it helps in reducing the power consumption due to the usage of sensors over time. The proposed method, in the considered test case and static environment, helped in reducing the power consumed by the sensors by about 40% when compared to the reference algorithm. In general, it is obvious that this power saving in dynamic environments definitely depends on the structure of the environment and its dynamicity. However, in environments with some dynamic variables in our test scenarios, the proposed methodology still proved to be effective, as utilizing it kept the power consumption about 50% less than if the sensors were used in a continuous mode.

In the future, we plan to further extend this work by increasing the depth of the swarm, i.e., the number of followers in the swarm, in order to further investigate the information exchange between the agents by taking the possible communication delays into consideration. Furthermore, the power consumption due to wireless communication and the effect of processing the information and coordinates on the battery life will also be analyzed. Because the leader undertakes all of the necessary computations, leading to faster draining of its battery and resultant shortening of overall life time of a mission, it will be interesting to analyze the effectiveness of an election-based leadership solution and explore different options for optimally swapping the leadership role amongst the followers, e.g., swap the places by selecting the follower with highest remaining battery life. Further, the effect of such improvements on the overall mission life needs to be studied. Additionally, finally moving onto testing the proposed approach in real-time initially under static environments and further moving to real-time controlled dynamic environments.

Author Contributions: Conceptualization, J.N.Y., M.-H.H., and M.M.Y.; methodology, J.N.Y., H.M., M.-H.H., and M.M.Y.; software, J.N.Y. and H.M.; validation, J.N.Y. and H.M.; writing-original draft preparation, J.N.Y., M.-H.H., M.M.Y., and J.P.; writing-review and editing, M.-H.H., M.M.Y., and J.P.; supervision, M.-H.H., M.M.Y. and J.P.; funding acquisition, M.-H.H. and J.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported in part by the Academy of Finland-funded research project 314048 and Nokia Foundation (Award No. 20200147).

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Campion, M.; Ranganathan, P.; Faruque, S. A Review and Future Directions of UAV Swarm Communication Architectures. In Proceedings of the 2018 IEEE International Conference on Electro/Information Technology (EIT), Rochester, MI, USA, 3–5 May 2018; pp. 903–908.
2. Shakhatreh, H.; Sawalmeh, A.H.; Al-Fuqaha, A.; Dou, Z.; Almaita, E.; Khalil, I.; Othman, N.S.; Khreishah, A.; Guizani, M. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access* **2019**, *7*, 48572–48634. [CrossRef]
3. Murray, R. Recent Research in Cooperative Control of Multi-Vehicle Systems. *J. Dyn. Syst. Meas. Control* **2007**, *129*, 571–598. [CrossRef]
4. He, L.; Bai, P.; Liang, X.; Zhang, J.; Wang, W. Feedback formation control of UAV swarm with multiple implicit leaders. *Aerosp. Sci. Technol.* **2018**, *72*, 327–334. [CrossRef]
5. Besada, J.A.; Bergesio, L.; Campaña, I.; Vaquero-Melchor, D.; López-Araquistain, J.; Bernardos, A.M.; Casar, J.R. Drone Mission Definition and Implementation for Automated Infrastructure Inspection Using Airborne Sensors. *Sensors* **2018**, *18*. [CrossRef] [PubMed]

6. Mualla, Y.; Najjar, A.; Daoud, A.; Galland, S.; Nicolle, C.; Yasar, A.U.H.; Shakshuki, E. Agent-based simulation of unmanned aerial vehicles in civilian applications: A systematic literature review and research directions. *Future Gener. Comput. Syst.* **2019**, *100*, 344–364. [CrossRef]
7. Gkiokas, A.; Cristea, A.I. Cognitive agents and machine learning by example: Representation with conceptual graphs. *Comput. Intell.* **2018**, *34*, 603–634. [CrossRef]
8. Dorri, A.; Kanhere, S.S.; Jurdak, R. Multi-Agent Systems: A Survey. *IEEE Access* **2018**, *6*, 28573–28593. [CrossRef]
9. Yasin, J.N.; Haghbayan, M.H.; Heikkonen, J.; Tenhunen, H.; Plosila, J. Formation Maintenance and Collision Avoidance in a Swarm of Drones. In Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control, ISCSIC 2019, Amsterdam, The Netherlands, 25–27 September 2019; Association for Computing Machinery: New York, NY, USA, 2019. [CrossRef]
10. Zhuge, C.; Cai, Y.; Tang, Z. A Novel Dynamic Obstacle Avoidance Algorithm Based on Collision Time Histogram. *Chin. J. Electron.* **2017**, *26*, 522–529. [CrossRef]
11. Wang, X.; Yadav, V.; Balakrishnan, S.N. Cooperative UAV Formation Flying With Obstacle/Collision Avoidance. *IEEE Trans. Control Syst. Technol.* **2007**, *15*, 672–679. [CrossRef]
12. Choi, J.; Kim, Y. Fuel-Efficient Formation Flight-Control Design Based on Energy Maneuverability. *J. Guid. Control Dyn.* **2008**, *31*, 1145–1150.
13. Lin, Y.; Saripalli, S. Collision avoidance for UAVs using reachable sets. In Proceedings of the 2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015, Denver, CO, USA, 9–12 June 2015; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2015; pp. 226–235. [CrossRef]
14. Esfahlani, S.S. Mixed reality and remote sensing application of unmanned aerial vehicle in fire and smoke detection. *J. Ind. Inf. Integr.* **2019**. [CrossRef]
15. Valavanis, K.P. *Unmanned Aircraft Systems: The Current State-Of-The-Art*; Springer: Berlin/Heidelberg, Germany, 2016.
16. Wargo, C.A.; Church, G.C.; Glaneueski, J.; Strout, M. Unmanned Aircraft Systems (UAS) research and future analysis. In Proceedings of the 2014 IEEE Aerospace Conference, Big Sky, MT, USA, 1–8 March 2014; pp. 1–16.
17. Yasin, J.N.; Mohamed, S.A.S.; Haghbayan, M.H.; Heikkonen, J.; Tenhunen, H.; Plosila, J. Navigation of Autonomous Swarm of Drones Using Translational Coordinates. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness*; The PAAMS Collection; Demazeau, Y., Holvoet, T., Corchado, J.M., Costantini, S., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 353–362.
18. Madridano, A.; Al-Kaff, A.; Martín, D.; de la Escalera, A. 3D Trajectory Planning Method for UAVs Swarm in Building Emergencies. *Sensors* **2020**, *20*, 642. [CrossRef]
19. Yasin, J.N.; Mohamed, S.A.S.; Haghbayan, M.; Heikkonen, J.; Tenhunen, H.; Plosila, J. Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches. *IEEE Access* **2020**, *8*, 105139–105155. [CrossRef]
20. Prats, X.; Delgado, L.; Ramirez, J.; Royo, P.; Pastor, E. Requirements, issues, and challenges for sense and avoid in unmanned aircraft systems. *J. Aircr.* **2012**, *49*, 677–687. [CrossRef]
21. Ferrera, E.; Alcántara, A.; Capitán, J.; Castaño, A.; Marrón, P.; Ollero, A. Decentralized 3D Collision Avoidance for Multiple UAVs in Outdoor Environments. *Sensors* **2018**, *18*, 4101. [CrossRef]
22. Yasin, J.N.; Mohamed, S.A.S.; Haghbayan, M.-H.; Heikkonen, J.; Tenhunen, H.; Yasin, M.M.; Plosila, J. Night vision obstacle detection and avoidance based on Bio-Inspired Vision Sensors. In Proceedings of the 2020 IEEE Sensors, Rotterdam, The Netherlands, 25–28 October 2020; pp. 1–4. [CrossRef]
23. Choi, D.; Lee, K.; Kim, D. Enhanced Potential Field-Based Collision Avoidance for Unmanned Aerial Vehicles in a Dynamic Environment. In Proceedings of the AIAA Scitech 2020 Forum, Orlando, FL, USA, 6–10 January 2020. [CrossRef]
24. Radmanesh, M.; Kumar, M.; Guentert, P.H.; Sarim, M. Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study. *Unmanned Syst.* **2018**, *6*, 95–118. [CrossRef]
25. Seo, J.; Kim, Y.; Kim, S.; Tsourdos, A. Collision Avoidance Strategies for Unmanned Aerial Vehicles in Formation Flight. *IEEE Trans. Aerosp. Electron. Syst.* **2017**, *53*, 2718–2734. [CrossRef]
26. Zhu, X.; Liang, Y.; Yan, M. A Flexible Collision Avoidance Strategy for the Formation of Multiple Unmanned Aerial Vehicles. *IEEE Access* **2019**, *7*, 140743–140754. [CrossRef]
27. Zhang, X.; Liniger, A.; Borrelli, F. Optimization-based collision avoidance. *arXiv* **2017**, arXiv:1711.03449.
28. Pham, H.; Smolka, S.A.; Stoller, S.D.; Phan, D.; Yang, J. A survey on unmanned aerial vehicle collision avoidance systems. *arXiv* **2015**, arXiv:1508.07723.
29. Smith, N.E.; Cobb, R.; Pierce, S.J.; Raska, V. Optimal collision avoidance trajectories via direct orthogonal collocation for unmanned/remotely piloted aircraft sense and avoid operations. In Proceedings of the AIAA Guidance, Navigation, and Control Conference, National Harbor, MD, USA, 13–17 January 2014; p. 966.
30. Yasin, J.N.; Mohamed, S.A.S.; Haghbayan, M.H.; Heikkonen, J.; Tenhunen, H.; Yasin, M.M.; Plosila, J. Energy-Efficient Formation Morphing for Collision Avoidance in a Swarm of Drones. *IEEE Access* **2020**, *8*, 170681–170695. [CrossRef]
31. Anderson, B.D.O.; Fidan, B.; Yu, C.; Walle, D. UAV Formation Control: Theory and Application. In *Recent Advances in Learning and Control*; Blondel, V.D., Boyd, S.P., Kimura, H., Eds.; Springer: London, UK, 2008; pp. 15–33.

32. Hoang, V.T.; Phung, M.D.; Dinh, T.H.; Ha, Q.P. Angle-Encoded Swarm Optimization for UAV Formation Path Planning. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 5239–5244.
33. Wu, X.; Wang, S.; Xing, M. Observer-Based Leader-Following Formation Control for Multi-Robot With Obstacle Avoidance. *IEEE Access* **2019**, *7*, 14791–14798. [CrossRef]
34. Ren, W. Consensus based formation control strategies for multi-vehicle systems. In Proceedings of the 2006 American Control Conference, Minneapolis, MN, USA, 14–16 June 2006; p. 6.
35. Low, C.B.; Ng, Q.S. A flexible virtual structure formation keeping control for fixed-wing UAVs. In Proceedings of the 2011 9th IEEE International Conference on Control and Automation (ICCA), Santiago, Chile , 19–21 December 2011; pp. 621–626.
36. Oh, K.K.; Park, M.C.; Ahn, H.S. A survey of multi-agent formation control. *Automatica* **2015**, *53*, 424–440. [CrossRef]
37. Shen, D.; Sun, Z.; Sun, W. Leader-follower formation control without leader's velocity information. *Sci. China Inf. Sci.* **2014**, *57*, 1–12. [CrossRef]
38. Han, Q.; Li, T.; Sun, S.; Villarrubia, G.; de la Prieta, F. "1-N" Leader-Follower Formation Control of Multiple Agents Based on Bearing-Only Observation. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability: The PAAMS Collection*; Demazeau, Y., Decker, K.S., Bajo Pérez, J., de la Prieta, F., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 120–130.
39. Yasin, J.N.; Haghbayan, M.H.; Yasin, M.M.; Plosila, J. Swarm Formation Morphing for Congestion Aware Collision Avoidance. *arXiv* **2020**, arXiv:cs.RO/2011.03883.
40. Bayezit, I.; Fidan, B. Distributed Cohesive Motion Control of Flight Vehicle Formations. *IEEE Trans. Ind. Electron.* **2013**, *60*, 5763–5772. [CrossRef]
41. Beard, R.W.; Lawton, J.; Hadaegh, F.Y. A coordination architecture for spacecraft formation control. *IEEE Trans. Control Syst. Technol.* **2001**, *9*, 777–790. [CrossRef]
42. Li, N.H.; Liu, H.H. Formation UAV flight control using virtual structure and motion synchronization. In Proceedings of the 2008 American Control Conference, IEEE, Seattle, WA, USA, 11–13 June 2008; pp. 1782–1787.
43. Dong, L.; Chen, Y.; Qu, X. Formation Control Strategy for Nonholonomic Intelligent Vehicles Based on Virtual Structure and Consensus Approach. Green Intelligent Transportation System and Safety. *Procedia Eng.* **2016**, *137*, 415–424. [CrossRef]
44. Lawton, J.R.; Beard, R.W.; Young, B.J. A decentralized approach to formation maneuvers. *IEEE Trans. Robot. Autom.* **2003**, *19*, 933–941. [CrossRef]
45. Balch, T.; Arkin, R.C. Behavior-based formation control for multirobot teams. *IEEE Trans. Robot. Autom.* **1998**, *14*, 926–939. [CrossRef]
46. Bartashevich, P.; Koerte, D.; Mostaghim, S. Energy-saving decision making for aerial swarms: PSO-based navigation in vector fields. In Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI), Honolulu, HI, USA, 27 November–1 December 2017; pp. 1–8. [CrossRef]
47. Majd, A.; Loni, M.; Sahebi, G.; Daneshtalab, M. Improving Motion Safety and Efficiency of Intelligent Autonomous Swarm of Drones. *Drones* **2020**, *4*, 48. [CrossRef]
48. Tseng, C.M.; Chau, C.K.; Elbassioni, K.M.; Khonji, M. Flight tour planning with recharging optimization for battery-operated autonomous drones. *arXiv* **2017**, arXiv:1703.10049.
49. Zorbas, D.; Razafindralambo, T.; Luigi, D.P.P.; Guerriero, F. Energy Efficient Mobile Target Tracking Using Flying Drones. *Procedia Comput. Sci.* **2013**, *19*, 80–87. [CrossRef]
50. Al-Sabban, W.H.; Gonzalez, L.F.; Smith, R.N. Wind-energy based path planning for Unmanned Aerial Vehicles using Markov Decision Processes. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6–10 May 2013; pp. 784–789. [CrossRef]
51. Puck LITE Datasheets-Velodynelidar-PDF Catalogs-Technical Documentation. Available online: https://pdf.directindustry.com/pdf/velodynelidar/puck-lite-datasheets/182407-676096.html (accessed on 16 February 2021)
52. Legacy XBee S1 802.15.4 Product Datasheet. Available online: https://www.digi.com/resources/library/data-sheets/ds_xbeemultipointmodules (accessed on 16 February 2021)
53. Zhu, A.Z.; Thakur, D.; Özaslan, T.; Pfrommer, B.; Kumar, V.; Daniilidis, K. The Multivehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception. *IEEE Robot. Autom. Lett.* **2018**, *3*, 2032–2039. [CrossRef]