Using Optimization, Learning, and Drone Reflexes to Maximize Safety of Swarms of Drones

Majd, Amin; Ashraf, Adnan; Troubitsyna, Elena; Daneshtalab, Masoud

Published in:
2018 IEEE Congress on Evolutionary Computation (CEC)

DOI:
10.1109/CEC.2018.8477920

Publicerad: 01/01/2018

Using Optimization, Learning, and Drone Reflexes to Maximize Safety of Swarms of Drones

Amin Majd*, Adnan Ashraf*, Elena Troubitsyna*, and Masoud Daneshtalab†
*Faculty of Natural Sciences and Technology, Åbo Akademi University, Turku, Finland
Email: amajd@abo.fi, aashraf@abo.fi, etroubit@abo.fi
†Division of Intelligent Future Technologies, Mälardalen University, Västerås, Sweden
Email: masoud.daneshtalab@mdh.se

***Abstract*—Despite the growing popularity of swarm-based applications of drones, there is still a lack of approaches to maximize the safety of swarms of drones by minimizing the risks of drone collisions. In this paper, we present an approach that uses optimization, learning, and automatic immediate responses (reflexes) of drones to ensure safe operations of swarms of drones. The proposed approach integrates a high-performance dynamic evolutionary algorithm and a reinforcement learning algorithm to generate safe and efficient drone routes and then augments the generated routes with dynamically computed drone reflexes to prevent collisions with unforeseen obstacles in the flying zone. We also present a parallel implementation of the proposed approach and evaluate it against two benchmarks. The results show that the proposed approach maximizes safety and generates highly efficient drone routes.**

***Index Terms*—Safety; path planning; drone; swarm; evolutionary algorithms; machine learning; reflexes**

I. Introduction

Drones are semi-autonomous aircrafts that can be controlled and operated remotely. Commercially-available drones are increasingly been used in a variety of applications such as monitoring and surveillance, search and rescue operations, photography and filming, and aerial package delivery. However, with the growing popularity and use of drones for consumer applications, the number of incidents involving drones is also increasing dramatically. In the United States alone, the Federal Aviation Administration receives more than 100 reports every month of unauthorized and potentially hazardous drone activity reported by pilots, citizens, and law enforcement[1]. Ensuring a hazard-free, safe flight is also equally important for indoor applications. Therefore, motion safety of drones is a prime concern for drone operators. It refers to the ability of the drones to detect and avoid collisions with static and moving obstacles in the flying zone. Moreover in scenarios involving a swarm or fleet of drones, motion safety also entails that the drones do not collide with one another.

In this paper, we present an approach that integrates optimization, learning, and automatic immediate responses (reflexes) of drones to generate efficient and safe routes for swarms of drones. We assume that the swarm executes certain missions, in which each drone flies from a start location to a destination location. The proposed approach uses geographical locations of the drones and of the successfully detected, static and dynamically appearing, moving obstacles to predict and avoid: (1) drone-to-drone collisions, (2) drone-to-static-obstacle collisions, and (3) drone-to-moving-obstacle collisions. The proposed approach comprises three main components: (1) a high-performance dynamic evolutionary algorithm (EA) for optimizing drone routes, (2) a reinforcement learning algorithm for incorporating the feedback and runtime data about the system state, and (3) a novel reactive module to dynamically compute drone reflexes to prevent collisions with unforeseen obstacles in the flying zone.

To find efficient drone routes, we propose a parallel and dynamic implementation of the imperialistic competition algorithm (ICA) [1] that allows us to find efficient collision-free routes for the drones in the swarm. The learning component is based on the K-nearest neighbor (KNN) learning algorithm [2]. Each placement produced by our parallel ICA for a given swarm and environment state is evaluated to train the system [3]. Both the learning and optimization algorithms work in parallel to compute alternative routing solutions. The results are compared and the most efficient solution is chosen. Since with each run the training set increases, eventually the learning algorithm becomes capable of proposing better solutions. To maximize safety, we augment the generated routes with dynamically computed drone reflexes. The drone reflexes computation module mimics a self-preservation control mechanism of humans. The reflexes are the automatic immediate or mechanical responses to particular hazardous situations, such as quickly moving the hand away from a hot surface. They aim at mitigating and confining the effects and damages of suddenly occurring hazards. In our proposed approach, when a drone detects a possible collision with an unforeseen obstacle, the drone reflexes computation module quickly computes a reflex movement for the drone to prevent and mitigate the collision. We also present a parallel implementation of the proposed approach and evaluate it against two benchmarks. The results show that the proposed approach produces highly efficient and safe routes.

The paper is structured as follows. Section II briefly reviews EAs and ICA. Section III presents the proposed approach that integrates optimization, learning, and drone reflexes. Section IV presents some important implementation details and experimental results. Finally in Section V, we review important related works and present our conclusions.

[1]https://www.faa.gov/uas/resources/uas_sightings_report/

II. Imperialistic Competition Algorithm

Evolutionary computing comprises a set of optimization algorithms, which are inspired by a biological or societal evolution [1]. The evolutionary algorithms (EAs) are widely used in the swarm systems due to their ability to find, in a highly performant way, near-optimal solutions for the computationally hard problems. An EA mimics the survival of the fittest principle of the nature.

Most EAs start from a random generation of the initial population of genotypes – the encodings of candidate solutions – in the overall search space according to a certain probability distribution. For each genotype, we can evaluate the fitness function, which represents the requirements to which the population should adapt. A fitness function assigns quality measures to the genotypes and drives the population improvement. The genotypes with the higher values of the fitness function get a higher probability to be chosen as the parents of the next generation. The chosen parents undergo variations to create offsprings. A variation consists of mutation and recombination. Mutation is a unary operator applied to a genome to produce a (slightly) modified mutant - a child (offspring). Mutation is stochastic, that is, the child depends on the outcomes of random choices. Recombination (or crossover) merges the information from two parent genotypes into offspring genotypes. Similarly to mutation, the recombination is also stochastic. Since the EAs keep the population size constant, we need to implement a survivor selection mechanism that chooses the individuals that remain in the next generation. Typically, it relies on the fitness ranking over the united set of parents and offsprings and selecting the top fittest segment as the next generation. The algorithm terminates either when the predefined number of generations have been produced, time allocated for running the algorithm has elapsed, or the successive generations bring only a negligible improvement.

There is a large variety of EAs. Some of them are inspired by natural phenomena, while others, such as Imperialist Competitive Algorithm (ICA) [1], by the social processes. The algorithm simulates a human social evolution. Its parallel implementation [4] has shown a remarkable performance in comparison with other EAs and offers a promising solution supporting compute-intensive tasks of swarm-based systems.

Figure 1 presents a flowchart highlighting the main steps in the ICA. The algorithm starts by a random generation of a set of countries - the genotypes - in the search space of the optimization problem. The fitness function determines the power of each country. The countries with the best values of the fitness function become *imperialists*, while the other countries become *colonies*. The colonies are divided among the imperialists and hence the overall search space is divided into empires. An association of a colony with an imperialist means that only the genotype of the imperialist and its associated colonies are used for crossover. The intuition behind it as follows: since the imperialist has a higher value of the fitness function, by crossing over with an associated colony, which is known to have a lower value of the fitness function, we inherit the strongest traits of the current population.

The mutation and crossover are implemented by *assimilation* and *revolution* operators. Assimilation moves colonies closer to an imperialist in its socio-political characteristics. For instance, it can be implemented by replacing a certain bit in a colony genotype with the corresponding bit of the imperialist. Revolution results in a drastic change of a colony's characteristics. It can be implemented by a random replacement of a certain bit in the colony genotype. As a result of assimilation and revolution, a colony might reach a better position and get a chance to take over the control of the entire empire, that is, to overthrow the current imperialist. This can happen only if the evaluation of the fitness function of such a colony gives a higher value (when solving a maximization problem) than the value of the fitness function of the current imperialist.

The next step of the algorithm computes the power of each empire and implements the imperialistic competition, which corresponds to the selection of the survivals process. The power of an empire is computed by aggregating the fitness value of the imperialist and a weighted sum of the fitness values of the colonies. The imperialists also try to take possession of colonies of other empires, that is, the weakest empire loses its weakest colony. In each step of the algorithm, based on their power, all the empires get a chance to take control of one or more of the colonies of the weakest empire. The steps of the algorithm are repeated until a termination condition is reached. As a result, the imperialist of the strongest empire produces the best solution. To improve performance of ICA, we introduce the notion of multi-population, that is, we divide the overall search space into multiple populations or clusters and perform a local search within each one of them. The best local solutions are then taken as input to perform the search in the entire search space. The multi-population based search also allows to use the inter-population *migration* operation, which migrates the best country from one population and uses it to replace the worst country in another population. Since the local search procedures are independent of each other, they can be implemented in parallel. Moreover, the multi-population based search enables a wider exploration of the search space, which helps to find high quality solutions.

III. The Proposed Approach

A swarm of drones is a typical example of a complex distributed networked system [5]. Each drone can be seen as a mobile sensing node that is capable of collecting monitoring data and communicating with some other drones in the swarm as well as with the cloud-based navigation center. Finding an efficient and safe route for each drone is a complex optimization problem. Therefore, we need to rely on certain heuristics to achieve the required objectives. In this paper, we use a dynamic EA to compute the drone routes [6]. The proposed algorithm is based on the imperialistic competition algorithm (ICA) [1].

Figure 2 presents an overview of our proposed approach called DIANA (Dynamic Intelligent Autonomous Navigation

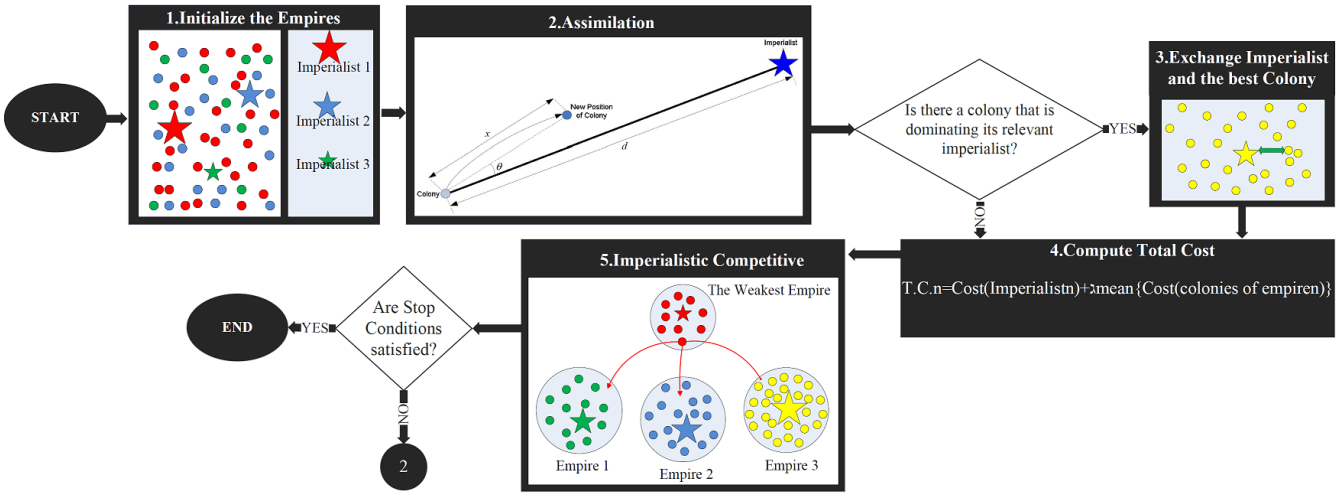

Fig. 1. A flowchart highlighting the main steps in the imperialistic competition algorithm

Algorithm) [3]. The *Offline Part* in the figure uses our proposed ICA-based route generation algorithm to generate drone routes before the start of the mission. Moreover, it computes and uses the shortest paths between the start and destination locations of the drones. Since a drone swarm is a highly dynamic system, we augment our offline module with an *online* approach that provides runtime means for monitoring and reconfiguration. The information obtained from the *Dynamic Monitoring* component has two main purposes. On one hand, it is a feedback mechanism. On the other hand, it allows us to detect the changes in the drone swarm and in the flying zone. Such changes may invoke swarm reconfiguration and regeneration of the drone routes. In addition, the *Prediction* module uses the runtime monitoring data to predict the movement of drones and moving obstacles in the flying zone. We feed the monitoring data and prediction results to both a *Dynamic EA* and a *Learning Algorithm*. Both modules compute the alternative routes. The routes are compared by the *Decision Center* that chooses the best solutions from the proposed alternatives. The *Navigation Center* then issues the corresponding commands to the drones. The *Safe Area Computation* and the *Reflexes Computation* modules compute the safe area and drone reflexes, respectively. They implement our proposed drone reflexes approach. The safe area computation module uses the information of the known and predicted obstacles to compute a safe area for each drone. Moreover, when a drone suddenly detects a possible collision with an unpredicted obstacle, the reflexes computation module quickly computes a reflex movement for the drone to prevent and mitigate the collision.

The proposed DIANA approach uses the parallel implementation of the ICA [4] with an integrated migration operation (referred henceforth as the MICAP algorithm). The MICAP algorithm computes the initial drone routes as well as the subsequent new routes during the execution of the mission. For the learning module, we use a K-nearest neighbor (KNN)-based learning algorithm [2] that runs alongside the MICAP algorithm.

The main steps of the DIANA approach are presented in Algorithm 1. The algorithm starts with the preplanning of the mission - the offline execution of MICAP to find the initial routes. It then obtains the current actual and predicted state of the system by invoking the dynamic monitoring and prediction modules and runs the MICAP algorithm and the KNN-based learning algorithm until the mission completes. In each iteration, it compares the results of MICAP with that of the KNN-based learning algorithm and selects the best solutions. The KNN algorithm is a popular learning method. In this paper, we use it for classification. The main idea of the algorithm is as follows. We select K samples in the training set. The algorithm predicts the numerical target - the nearest neighbor - based on a similarity measure, which in our case

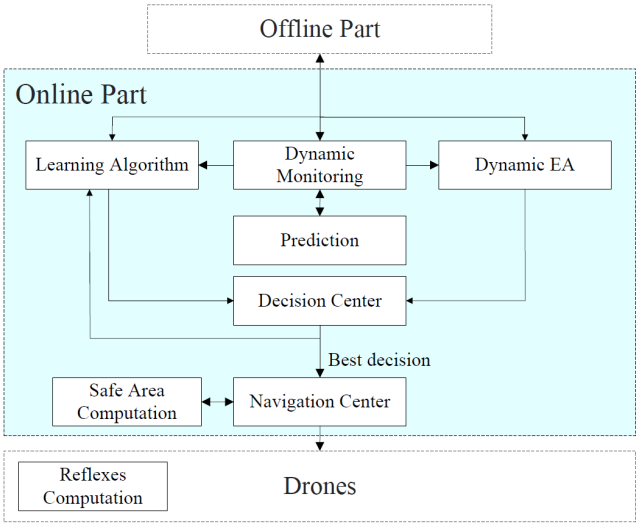

Fig. 2. The proposed DIANA approach

is a distance function. We compute the numerical target as the average of the Euclidian distances of the *K* nearest neighbors.

In our approach, the algorithm takes as the input *K* solutions generated for the same system state. Then it computes an average solution. For each drone in the swarm, it takes *K* routes proposed for it. Then it computes an average route in terms of the Euclidian distance. By computing such a route for each drone in the swarm, the learning component calculates the complete solution for the swarm. The decision center uses a fitness function to compute fitness values for the solutions produced by MICAP and the learning algorithm. It then chooses the solution with the highest fitness value.

The prediction module implements a two-step approach [7], [8], which allows to make predictions under real-time constraints. The approach involves *trackers* that offer a representative view of the trends to the *predictors*, thus achieving the two-step approach. In a three-dimensional flying zone, the current location of an obstacle or a drone monitored at time t_i contains three values (x, y, z), which represent the three axis. Therefore, for predicting the future location of an obstacle or a drone at time t_{i+k}, we use three parallel trackers and three parallel predictors. Our prediction approach is further elaborated in [3].

The computation of the drone reflexes is elaborated in the example depicted in Figure 3 and 4. Figure 3 shows the safe area for Drone1. It shows the radius of the safe area (r=4), the repulsive forces of Drone2 and Drone3 to Drone1 (F2 and F3), and the total force direction (FD). In each iteration, our algorithm computes FD based on all drones in the safe area. The computation of the repulsive forces is based on the distances of Drone2 and Drone3 from Drone1 (Dis2 and Dis3). In this example, Dis2 and Dis3 are $\sqrt{8}$ and 3, respectively. In each iteration, the repulsive forces and FD are computed in the online part in the cloud layer (as shown in Algorithm 1). For example to prevent Drone1 from colliding with Drone2, the repulsive force for Drone1 is computed as $F2 = (r - Dis2)$ along with the reflex direction in the opposite direction of Drone2.

Figure 4 illustrates that when Drone1 detects an unpredicted obstacle, it computes its reflex position based on the total force direction (FD) and the force of the obstacle to Drone1 (FOb). FOb is a unit vector in the opposite direction of the obstacle. Finally, the drone computes the safe direction (SD) for reflex movement. SD is a vector from the drone to a safe position in the flying zone. It is computed as the sum of FD and FOb.

The pseudo-codes of the proposed MICAP, KNN, safe area computation, and drone reflex computation algorithms are presented in Procedure 1, 2, 3, and 4, respectively. All four procedures are invoked by the main DIANA algorithm presented in Algorithm 1. The MICAP, KNN, and safe area computation modules can be implemented on a remote server in the cloud layer, while the latency-sensitive operation involving the computation of drone reflexes runs in the drones layer or on a fog or edge server. Section IV elaborates our optimization approach and some important implementation details of DIANA.

Algorithm 1 DIANA

1: {**Offline part in the cloud layer**}
2: Best-Placement ← Call MICAP(Initial-State)
3: Send(Best-Placement) → Navigation Center
4: {**Online part in the cloud layer**}
5: **while** Mission is in progress **do**
6: Current-State ← Call Dynamic Monitoring and Prediction modules
7: Best-EC-Result ← Call MICAP(Current-State)
8: Best-KNN-Result ← Call KNN(Current-State)
9: Best-Placement ← Max(Best-EC-Result, Best-KNN-Result)
10: Send(Best-Placement) → Navigation Center
11: Send(Best-Placement)→ KNN-Dataset
12: FD ← Call Compute-Safe-Area(Current-State)
13: {**Online part in the drones layer**}
14: **if** a drone detects an unpredicted obstacle in its safe area **then**
15: Drone-Reflex-Position ← Call Compute-Drone-Reflex(Unpredicted-Obstacle-Position, FD)
16: Prevent and mitigate the collision by moving the drone to Drone-Reflex-Position
17: **end if**
18: **end while**

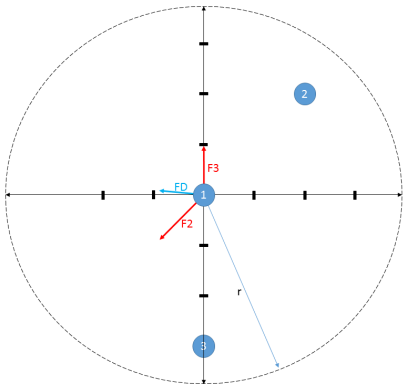

Fig. 3. First part of drone reflex computation

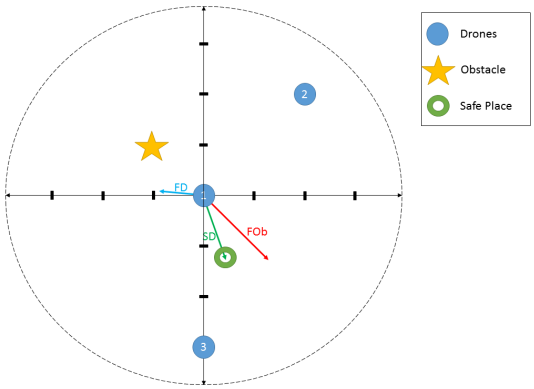

Fig. 4. Second part of drone reflex computation

Procedure 1 MICAP(Current-State)

if Processor P_i **then**
  for j=1 to Population Size **do**
    Initial-Population [j] $\leftarrow Country_j$
    Cost_Initial-Population[j] $\leftarrow \alpha \cdot$ Safety-Level + $\frac{\beta}{\text{Path-Length}}$
  end for
  Initial-Population $\leftarrow$ Sort Initial-Population
  for j=1 to #Empire **do**
    Imperialist[j] $\leftarrow$ Initial-Population [j]
  end for
  for j=#Empire+1 to Population Size **do**
    Assigned as a Colony to an Empire
  end for
  for All Colony **do**
    Assimilate Colony $\rightarrow$ Imperialist
  end for
  for All Colony **do**
    if Colony $\geq$ Imperialist **then**
      Colony $\leftrightarrow$ Imperialist
    end if
  end for
  for j=1 to #Empire **do**
    Empire Power[j] $\leftarrow$ Cost(Imperialist$_j$) + $\xi \cdot$ Mean(Cost(Colonies of Empire$_j$))
  end for
  Temp $\leftarrow$ Worst Country$_{\text{Worst Empire}}$
  Empire$_{l \neq \text{Worst Empire}} \leftarrow$ Temp
  if #Iterations % Migration Gap=0 **then**
    Best Country $\leftarrow$ Worst Country$_{\text{Processor P}_{(i+1)\%\#\text{Processors}}}$
    Worst Country $\leftarrow$ Best Country$_{\text{Processor P}_{(i+1)\%\#\text{Processors}}}$
  end if
  if termination condition **then**
    return Best Country
  end if
end if

Procedure 2 KNN(Current-State)

for All Instance $\in$ Dataset **do**
  Find Nearest-Neighbor
end for
return Nearest-Neighbor

Procedure 3 Compute-Safe-Area(Current-State)

for i in 1 to number of drones **do**
  Ds$_i \leftarrow$ (Drones in safe area $\cup$ Obstacles in safe area)
  for j in 1 to |Ds$_i$| **do**
    FD$_i$= $\sum_{k=1}^{o}$ (F$_{O_j \rightarrow D_i}$ $\cdot$ (r - Dis$_{D_i \rightarrow O_j}$))
  end for
end for
return FD

Procedure 4 Compute-Drone-Reflex

FOb $\leftarrow F_{O_{\text{unpredicted\_obstacles}} \rightarrow D_i} \cdot (r - Dis_{D_i \rightarrow O_{\text{unpredicted\_obstacles}}})$
SD $\leftarrow$ FOb + FD$_i$
return SD

IV. Implementation and Experimental Results

In this section, we present some important implementation details and experimental results. We assume that the flying zone $AREA$ is represented by a three-dimensional grid. However, for a simpler illustration, we describe the main steps with a two-dimensional grid shown in Figure 5(a). Our goal is to find an efficient, collision-free route for each drone from the initial start location of the drone to the destination location.

We use the example shown in Figure 5(b) to explain the principles used for defining the countries in the MICAP algorithm. For the drone $d1$, initially situated at location 20, the shortest path from the initial location to the destination is a sequence $\langle 20, 19, 18, 17, 16, 11, 6\rangle$. We note that the path of each drone can be succinctly represented by a *turning* point – we call it *middle point*, which would be 16 for $d1$. Therefore, the proposed approach uses the middle points to optimize the drone routes. By using different values for a drone's middle point, it is possible to generate different routes for the drone. Thus, it allows to explore different alternatives in the search space. Lets assume that the middle points for the two other drones in Figure 5(b) are 12 and 9. A country representing all drone routes for the swarm in Figure 5 can then be defined as a triple $\langle\langle 16, 12, 9\rangle\rangle$. In general, for nd drones a country is an nd-tuple consisting of the middle points of the corresponding drones.

In the offline, planned part of the proposed approach, all locations occupied by static obstacles and the initial locations of the moving obstacles are marked as occupied or unsafe. The offline route planning module then generates all shortest paths between each pair of locations in the flying zone $AREA$ while avoiding all locations marked as occupied or unsafe and stores the generated routes in a database, which is then used to compose the routes for the individual drones as a concatenation of the shortest routes from initial locations to the middle points and from the middle points to the final destinations. The shortest routes are computed using the algorithm proposed by Dijkstra [9].

The fitness function to evaluate the fitness of each country optimizes the safety/performance ratio. The first argument of our fitness function is the distance metric

$$Distance\ Metric\ = \sum_{i=1}^{nd} Distance_{Current_i \rightarrow Middle_i} + Distance_{Middle_i \rightarrow Destination_i}$$

It defines the total length of the drone routes according to the given solution. For our example in Figure 5(b) the distance metric of the routing defined by the country $\langle 16, 12, 9\rangle$ is the sum of the lengths of the drone paths: 6+6+6=18. The second argument of the fitness function defines the number of cross points associated with the given solution. For our example in Figure 5(b), the number of cross points is 3: in location 17 between the route 1 and 2, in location 12 between the

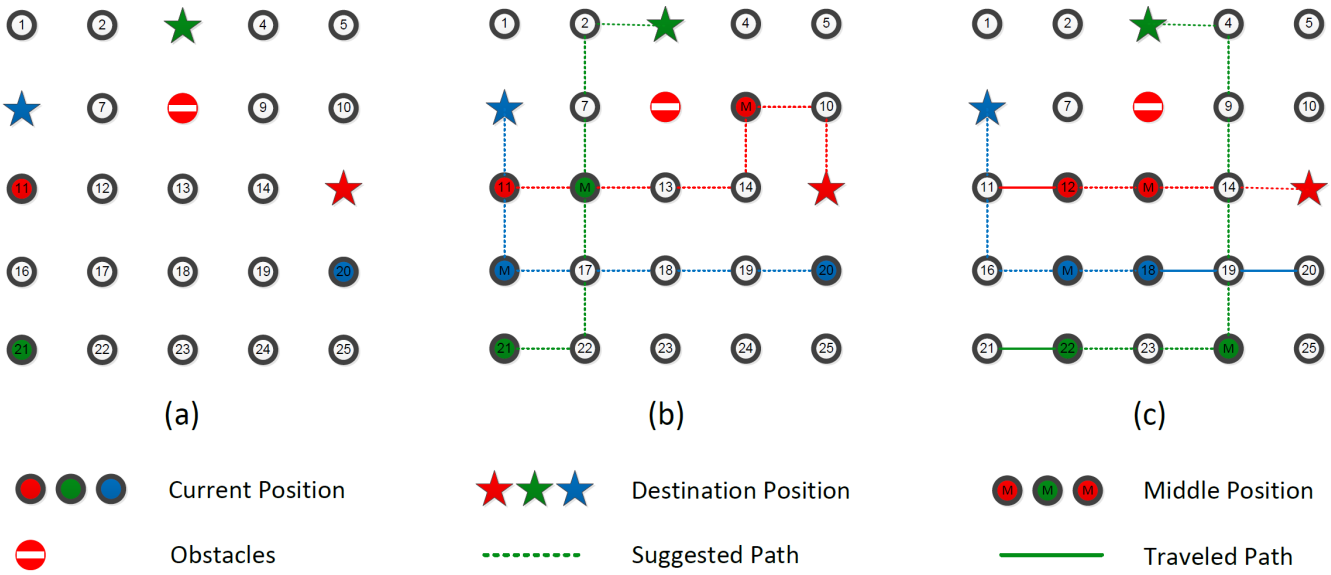

Fig. 5. An illustrative example of drone route planning

route 2 and 3, and in the location 11 between route 1 and 3, correspondingly.

The third argument is the safety level of the time gap at the cross point. We introduce three safety levels: 0 if there is no cross points, 1 if the time gap at the cross point is above the safety threshold, and 2 if the time gap is below the threshold. For our example in Figure 5(b), the time gap at cross point 17 is 1, because the drones arrive at that point at times 3 and 2, the time gap for the cross point 12 is 2, because the drones arrive there at times 3 and 1, and for the cross point 11, it is 5. As a matter of illustration, we can assume that the time gaps below threshold 2 are classified as level 2, while the time gaps at and above threshold 2 as level 1. Hence, the cross point 17 obtains level 2, while the cross points 11 and 12 obtain level 1 each. The safety level of a complete routing solution for the swarm can then be computed by aggregating the individual safety levels of all cross points in the solution: 2+1+1=4. We define our route optimization task as a minimization problem with the following fitness function:

$$Fitness\ Function\ =\ Distance\ Metric\ +\alpha\cdot Number\ of\ CrossPoint\ +\ \beta\cdot Level$$

Here α and β are the weight coefficients defined as

$$1\ \leq\alpha\leq\frac{nd}{2}\qquad\qquad 1\leq\beta\leq\sqrt{np}\times nd$$

where nd is the number of drones and np is the total number of points. These values allow us to adapt the fitness function evaluation based on the level of complexity of the flying zone and the number of drones. For our example in Figure 5(b), the value of the fitness function is computed as $18+1.5\times 3+5\times 4=42.5$.

A. Experiment Design and Setup

We have implemented the proposed DIANA approach on a shared memory model. We used the message passing interface (MPI) to parallelize the proposed algorithm and MPICH[2] to run the algorithm. To implement DIANA, we used four processors in a ring topology. Our algorithm was tested on Intel® Xeon® E5-1620 v3 @ 3.50 GHz processors with 16 GB memory and NVIDIA® GeForce® GTX 1080 graphics processing units.

We present results from two benchmark implementations. Benchmark 1 is based on a small 100×100×20 flying zone with 16 drones, 3 dynamic obstacles moving on straight lines from different starting positions, 2 dynamic obstacles moving randomly from different starting positions, and 8 unforeseen/unpredicted static obstacles. Benchmark 2 is based on a large 1000×1000×100 flying zone with 250 drones, 10 dynamic obstacles moving on straight lines from different starting positions, 15 dynamic obstacles moving randomly from different starting positions, and 60 unpredicted static obstacles. Figure 6 and 7 depict the routes of the moving obstacles in Benchmark 1 and 2, respectively. The experimental design is summarized in Table I.

We also compare the results with two alternative approaches:

- A well-known approach called Dynamic Genetic Algorithm (DGA) [10], which addresses a similar problem.
- A baseline approach called DANA (Dynamic Autonomous Navigation Algorithm), which is similar to DIANA except that it does not use learning and prediction.

B. Results and Analysis

Table II and III present a comparison of the results of DIANA, DGA, and DANA. The comparison is based on the following seven metrics:

[2]https://www.mpich.org/

TABLE I
EXPERIMENTAL DESIGN

	Benchmark 1	Benchmark 2
Flying zone	100×100×20	1000×1000×100
Problem size	Small	Large
Number of drones	16	250
Number of unpredicted static obstacles	8	60
Number of moving obstacles	5	25

1) *Route length:* the length of a drone route measured as the number of steps in the drone route. To be minimized to generate shorter routes.
2) *Minimum distance:* the minimum distance between a drone and an obstacle. To be maximized to generate safer routes.
3) *Frequency of route regeneration:* the number of times the drone routes are regenerated. To be minimized for reducing the re-computation overhead.
4) *Number of crashes:* the number of drone collisions. To be minimized to generate safer routes.
5) *Length of the longest route:* the total number of steps in the generated longest route. To be minimized to generate shorter routes.
6) *Total time:* total runtime of the algorithm in milliseconds. To be minimized for reducing the algorithm runtime.
7) *Time per step:* computation time for one step in milliseconds. To be minimized for reducing the computation overhead.

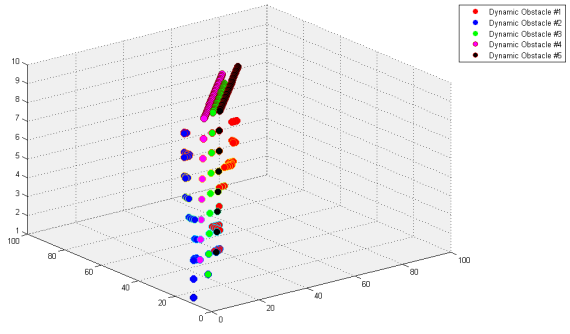

Fig. 6. Moving obstacles in Benchmark 1

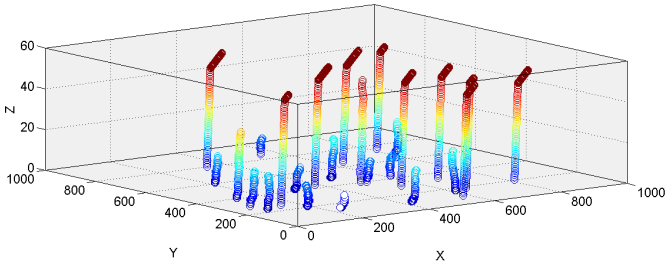

Fig. 7. Moving obstacles in Benchmark 2

TABLE III
A COMPARISON OF THE ALGORITHM RUNTIME FROM BENCHMARK 2

	DIANA	DANA	DGA
Total time (milliseconds)	**248565**	824015	4832700
Time per step (milliseconds)	**99.986**	326.731	1692.122

The results show that DIANA outperformed DGA and DANA in all seven metrics. It minimized the route lengths and the frequency of route regeneration more efficiently than DGA and DANA. Moreover, it generated drone routes with a minimum distance of 1 step, which ensured the safety of the drones. As a result, there were no drone collisions or crashes. In Benchmark 1, DIANA produced 14% and 16% shorter routes than DANA and DGA, respectively. Similarly, in Benchmark 2, it generated 15% and 13% shorter routes than DANA and DGA, respectively. DIANA also minimized the frequency of route regeneration, length of the longest route, total time, and time per step more efficiently than DANA and DGA. Therefore, DIANA generated safer and shorter drone routes while minimizing the computation overhead.

V. RELATED WORK AND CONCLUSIONS

The problem of motion safety of semi-autonomous robotic systems is currently attracting significant research attention. A comprehensive overview of the problems associated with autonomous mobile robots is given in [11]. The analysis carried out in [12] shows that the most prominent routing schemes do not guarantee motion safety. Our approach resolves this issue and ensures not only safety but also provides efficient online routing.

Macek et al. [13] proposed a layered architectural solution for robot navigation. However, in their work, they focused on the safety issues associated with the navigation of a single vehicle and did not consider the problem of collision-free navigation in the context of swarms of robots. Aniculaesei et al. [14] presented a formal approach that employs formal verification to ensure motion safety. Petti and Fraichard [15] proposed an approach that relies on partial motion planning to ensure safety. Their solution supports navigation of a single vehicle. In our work, we have discretized the flying zone and have developed a highly efficient approach that computes the next safe states for an entire swarm and provides a mechanism for online route regeneration and collision avoidance.

Olivieri and Endler [16] presented an approach for movement coordination of swarms of drones using smart phones and mobile communication networks. Their work focuses on the internal communication of the swarm and does not provide a solution for collision-free route generation. Barry and Tedrake [17] proposed an obstacle detection algorithm for drones that allows to detect and avoid collisions in realtime. Similarly, Lin [18] presented a realtime path planner for drones that detects and avoids moving obstacles. These approaches are only applicable for individual drones and they do not provide support for a swarm of drones. In our work, we focused on

TABLE II
A COMPARISON OF THE PROPOSED APPROACH WITH TWO ALTERNATIVE APPROACHES

	Benchmark 1			Benchmark 2		
	DIANA	DANA	DGA	DIANA	DANA	DGA
Route length	**3510**	4086	4203	**464486**	546788	532451
Minimum distance	**1**	0	0	**1**	0	0
Frequency of route regeneration	**8**	23	31	**331**	744	534
Number of crashes	**0**	5	5	**0**	18	48
Length of the longest route	**252**	272	266	**2486**	2522	2856

collision prediction and avoidance and efficient navigation of swarms of drones.

A comprehensive literature review on motion planning algorithms for drones can be found in [19]. The approaches reviewed in [19] are applicable to a preliminary, offline motion planning phase to plan and produce an efficient path or trajectory for a drone before the start of the mission. A more recent survey on motion planning of drones can be found in [20]. Augugliaro et al. [21] also presented a planned approach for generating collision-free trajectories for a drone fleet. In contrast to these approaches, our proposed approach combines offline motion planning with a more realistic online route generation approach to produce efficient collision-free routes.

In this paper, we presented an approach that uses optimization, learning, and automatic immediate responses (reflexes) of drones to ensure safe and efficient operations of swarms of drones. The proposed approach integrates a high-performance dynamic evolutionary algorithm and a reinforcement learning algorithm to generate safe and efficient drone routes and then augments the generated routes with dynamically computed drone reflexes to prevent collisions with unforeseen obstacles in the flying zone. We also presented a parallel implementation of the proposed approach and evaluated it against two benchmarks. The results showed that the proposed approach maximizes safety, generates highly efficient drone routes, and has a low computation overhead.

ACKNOWLEDGMENT

The work was supported by the Academy of Finland projects OpenCPS: Open Integrated Framework for Accelerating Development of Resilient CPS and CoRA: Continuous Resilience Assurance of Complex Software-Intensive Systems.

REFERENCES

[1] E. Atashpaz-Gargari and C. Lucas, "Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 4661–4667.

[2] M. Mejdoub and C. Ben Amar, "Classification improvement of local feature vectors over the KNN algorithm," *Multimedia Tools and Applications*, vol. 64, no. 1, pp. 197–218, 2013.

[3] A. Majd, A. Ashraf, E. Troubitsyna, and M. Daneshtalab, "Integrating learning, optimization, and prediction for efficient navigation of swarms of drones," in *Proceedings of the 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2018.

[4] K. Kar and S. Banerjee, "Node placement for connected coverage in sensor networks," WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2003. [Online]. Available: https://hal.inria.fr/inria-00466114

[5] N. Heo and P. K. Varshney, "Energy-efficient deployment of intelligent mobile sensor networks," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 35, no. 1, pp. 78–92, 2005.

[6] A. Majd, G. Sahebi, M. Daneshtalab, J. Plosila, and H. Tenhunen, "Placement of smart mobile access points in wireless sensor networks and cyber-physical systems using fog computing," in *IEEE International Conference on Scalable Computing and Communications (ScalCom)*, July 2016, pp. 680–689.

[7] M. Andreolini, S. Casolari, and M. Colajanni, "Models and framework for supporting runtime decisions in web-based systems," *ACM Transactions on the Web*, vol. 2, no. 3, pp. 17:1–17:43, 2008.

[8] A. Ashraf, B. Byholm, and I. Porres, "A session-based adaptive admission control approach for virtualized application servers," in *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, 2012, pp. 65–72.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[10] S. Indhumathi and D. Venkatesan, "Improving coverage deployment for dynamic nodes using genetic algorithm in wireless sensor networks," *Indian Journal of Science and Technology*, vol. 8, no. 16, 2015.

[11] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, ser. Intelligent robotics and autonomous agents. MIT Press, 2011.

[12] T. Fraichard, "A short paper about motion safety," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 1140–1145.

[13] K. Macek, D. Vasquez, T. Fraichard, and R. Siegwart, "Safe vehicle navigation in dynamic urban scenarios," in *2008 11th International IEEE Conference on Intelligent Transportation Systems*, 2008, pp. 482–489.

[14] A. Aniculaesei, D. Arnsberger, F. Howar, and A. Rausch, "Towards the verification of safety-critical autonomous systems in dynamic environments," in *Proceedings of the The First Workshop on Verification and Validation of Cyber-Physical Systems*, 2016, pp. 79–90.

[15] S. Petti and T. Fraichard, "Partial motion planning framework for reactive planning within dynamic environments," in *Proceedings of the IFAC/AAAI International Conference on Informatics in Control, Automation and Robotics*, 2005.

[16] B. J. O. de Souza and M. Endler, "Coordinating movement within swarms of UAVs through mobile networks," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, 2015, pp. 154–159.

[17] A. J. Barry and R. Tedrake, "Pushbroom stereo for high-speed navigation in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3046–3052.

[18] Y. Lin, "Moving obstacle avoidance for unmanned aerial vehicles," Ph.D. dissertation, Arizona State University, 2015.

[19] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous UAV guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1, p. 65, 2009.

[20] F. Kendoul, "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *Journal of Field Robotics*, vol. 29, no. 2, pp. 315–378, 2012.

[21] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1917–1922, 2012.