Evolutionary Optimization of Drone Trajectories Based on Optimal Reciprocal Collision Avoidance

Alex Bojeri, Giovanni Iacca
University of Trento
Trento, Italy
alex.bojeri@studenti.unitn.it, giovanni.iacca@unitn.it

***Abstract*—In recent years, the advent of new hardware and software technologies for navigation and control has made Unmanned Aerial Vehicles (UAVs) ever more autonomous and efficient. As a consequence, it is now possible to have drones moving within complex environments, such as cities or indoor areas. One of the main requirements for intelligent mission planning in such environments is the ability to correctly and efficiently detect and avoid obstacles. For this reason, various libraries have been created for the simulation of UAV navigation in virtual environments, in order to test algorithms for automatic obstacle detection and collision avoidance before deploying the drones in the real world. Usually, the performance of these algorithms depends on various parameters as well as specific application settings. However, while different parameter configurations can be easily tested in simulation, their number can be too large to allow a complete exploration of the parameter space or a manual tuning. Furthermore, a full analytical model of the parameters' influence on the algorithmic performance can be hard to obtain. Yet, it is extremely important to find their optimal values to allow collision-free navigation. In this direction, we propose here a thorough exploration, based on an Evolutionary Algorithm (EA), of the parameter space of the Optimal Reciprocal Collision Avoidance (ORCA) algorithm. Our results show that the proposed EA is a viable solution for finding optimal parameter settings that can be generalizable to different scenarios characterized by different complexity levels.**

I. Introduction

One of the most important aspects in Unmanned Aerial Vehicle (UAV) applications is path planning, i.e. the generation of a trajectory between the initial drone's location and its desired destination. In general, path planning can be formulated as a *trajectory optimization* problem, where the goal is to find an optimal trajectory w.r.t. one or more application-specific metrics, e.g. accuracy to reach the target position, obstacle/collision avoidance, energy consumption, etc., subject to physical constraints related to the environment and the properties of the drones themselves.

As highlighted in [1], searching for optimal paths is a difficult problem since: a) in most cases the search itself requires an excessive computational time and b) in some cases even the computation of just one path can be a rather complex task. These difficulties have been further exacerbated by the fact that with the constantly reducing cost and volume of drone hardware, coupled with improved performances in terms of e.g. speed/acceleration, drone applications can now be characterized by a large number of small-scale, highly dynamic drones which might be deployed in cluttered environments. In these scenarios, trajectory optimization becomes even harder, for instance because the number of possible collisions increases with the number of drones, and because avoiding collisions becomes harder as drones move faster.

Among the various optimization techniques available in the literature, Evolutionary Algorithms (EAs) [2] and other bio-inspired algorithms have shown a great potential for solving trajectory optimization problems. In fact, these algorithms allow to optimize a given objective function (in the evolutionary jargon, called "fitness" function) in a black-box manner: they only require to evaluate the objective function on any new candidate solution generated during the search, without using any gradient information. As such, EAs are gradient-free (zeroth-order) optimizers which can be used as metaheuristics for solving global optimization problems when gradient information is unknown or a mathematical formulation of the objective function is not even available. Furthermore, EAs are deemed generally quite robust in terms of convergence and optimization performance, and their complexity is affordable.

In recent years, bio-inspired computation –and especially EAs– have been applied to various aspects of UAVs [3], including path planning. For instance, in [4], an offline/online path planner for UAV navigation was presented, where an EA optimizes a curved path line in a 3-dimensional setting. The same problem, yet with a multi-objective optimization approach based on NSGA-II [5] configured for minimizing the trajectory length while maximizing the margin of safety, was proposed in [6]. Multi-objective problem formulations of trajectory optimization were also considered in [7], [8], and [9]. Single objective trajectory optimization was instead tackled in [10] and [11], where the authors used Differential Evolution to optimize 2-D trajectories described by successive way-points (in [10]) or B-spline curves (in [11]) in order to allow the UAVs to reach their destination while avoiding collisions. In [12], an EA was used to optimize navigation in a field of obstacles at uncertain locations. An EA hybridized with Particle Swarm Optimization (PSO) was instead used in [13], to solve a multi-UAV formation reconfiguration problem. More recently, as an alternative to EAs Multi-colony Ant Colony Optimization (Multi-ACO) was applied to trajectory optimization in [14]. Another recent study [15] considered the energy needed by drones to communicate with a ground terminal as primary objective for the evolutionary optimization of the UAV trajectories.

ISSN 2305-7254

One common aspect of these studies is that they usually tackle the trajectory optimization problem as the problem of searching for the optimal parameters *describing* the trajectories themselves, such as way-points or B-spline curves, rather than optimizing the way those trajectories are actually *generated*. This is the focus of the present work: here, we tackle a rather different problem since we assume that the trajectories are generated by means of the well-established Optimal Reciprocal Collision Avoidance (ORCA) algorithm [16], and our goal is to find the optimal parameters of ORCA itself. In fact, the performance of ORCA depends on a number of parameters which make manual tuning a rather hard and tedious task. Yet, to the best of our knowledge a systematic exploration of the ORCA parameter space has not been conducted so far, and EAs might represent a valuable tool for performing this parameter search.

In order to achieve our goal, we use a customizable EA available in the Python library *inspyred*[1] and we couple it with a collision avoidance simulator, namely the Reciprocal Velocity Obstacle (RVO2) C++ library [17][2], and its related Python bindings[3]. To apply the EA to the specific trajectory optimization problem, we define a specific fitness function by taking into account various aspects of the simulations, including the number of collisions and the accuracy to reach the target, and then we evaluate a large number of parameter ORCA combinations by measuring their effect on the *RVO2* simulation. To summarize, our main contributions are:

- We implement a fully customizable, open-source Python pipeline coupling an EA with the *RVO2* simulator[4].
- We conduct a systematic exploration of the ORCA parameter space in four different scenarios characterized by different environments and numbers of obstacles.
- We investigate the generalization capability of the optimal parameters from one scenario to the other.

The rest of the paper is structured as follows. In Section II, we introduce the background concepts regarding collision avoidance and EAs. In Section III, we describe the implementation details of the simulator and the EA configuration. Finally, in Section IV and Section V we present our experimental results and give the conclusions of this work.

II. Background

As we introduced in the previous section, our goal is to optimize the parameters of the Optimal Reciprocal Collision Avoidance (ORCA) algorithm [16]. ORCA consists in the management of n agents, each of which moves in a given space taking into account the other $n-1$ agents, estimating at each timestep their velocities to avoid collisions. Of note, the calculation of the velocities does not assume any type of communication between the n agents involved. Instead, it considers the position of each drone and the respective velocity for the definition of the set of velocities that allow avoidance of collisions.

ORCA is nowadays widely adopted in various applications, not only in drone navigation but also in robotics [18], [19] and multi-agent navigation for video games [20]. In [21], a 3-dimensional improved version of ORCA was presented, which includes the possibility to handle dynamic constraints. Another modified version of ORCA has been proposed in [22], which works also with speed constrained aircrafts. One particularly interesting application of ORCA has been recently proposed in [23], where a modified version of ORCA was applied in simulation to a large swarm of drones for continuous surveillance of an urban environment.

In the following, we briefly summarize the formal definition of Reciprocal Collision Avoidance and how this is applied in the ORCA algorithm.

A. Reciprocal Collision Avoidance

Reciprocal Collision Avoidance is defined as follows. Given two agents, A and B, the set of velocities $VO^{\tau}_{A|B}$ is defined as the set of relative velocities of A compared to B such that a collision between A and B will result before a certain time instant τ [24]. By introducing the following notation:

$$D(\mathbf{p}, r) = \{\mathbf{q} | \|\mathbf{q} - \mathbf{p}\| < r\} \tag{1}$$

in which D represents a circular space with origin $\mathbf{p}$ and radius r, $VO^{\tau}_{A|B}$ can be written as:

$$VO^{\tau}_{A|B} = \{\mathbf{v} | \exists t \in [0, \tau] :: t\mathbf{v} \in D\left(\mathbf{p}_B - \mathbf{p}_A, r_A + r_B\right)\}. \tag{2}$$

A graphical representation of $VO^{\tau}_{A|B}$ is shown in Fig. 1.

Given the velocities of the agents A and B, respectively $\mathbf{v}_A$ and $\mathbf{v}_B$, the goal of Reciprocal Collision Avoidance is to find a solution $\{\mathbf{v}_A, \mathbf{v}_B\} \notin VO^{\tau}_{A|B}$ that guarantees collision-free navigation between A and B at least until time τ.

In order to do that, considering a generalization of $VO^{\tau}_{A|B}$ according to the Minkowski sum, it is possible to define the set of velocities that guarantee the collision avoidance $CA^{\tau}_{A|B}(\mathbf{v}_B)$ for the agent A with respect to the agent B, given $\mathbf{v}_B$, as:

$$CA^{\tau}_{A|B}(\mathbf{v}_B) = \left\{\mathbf{v} | \mathbf{v} \notin VO^{\tau}_{A|B} \oplus \mathbf{v}_B\right\}, \tag{3}$$

thus deducing the respective velocity of mutual collision avoidance as: $\mathbf{v}_A \subseteq CA^{\tau}_{A|B}(\mathbf{v}_B)$ and $\mathbf{v}_B \subseteq CA^{\tau}_{B|A}(\mathbf{v}_A)$.

B. Optimal Reciprocal Collision Avoidance

Among all the possible pairs of values for the velocities $\mathbf{v}_A$ and $\mathbf{v}_B$, respectively in $CA^{\tau}_{A|B}(\mathbf{v}_B)$ and $CA^{\tau}_{B|A}(\mathbf{v}_A)$, ORCA seeks to find the closest ones to optimal velocities $\mathbf{v}^{\text{opt}}_A$ and $\mathbf{v}^{\text{opt}}_B$ [16]. Denoted $ORCA^{\tau}_{A|B}$ such velocity for A and $ORCA^{\tau}_{B|A}$ for B, it results that:

$$ORCA^{\tau}_{A|B} = \left\{\mathbf{v} | \left(\mathbf{v} - \left(\mathbf{v}^{\text{opt}}_A + \frac{1}{2}\mathbf{u}\right)\right) \cdot \mathbf{n} \geq 0\right\}, \tag{4}$$

where $\mathbf{n}$ is the normal direction at the boundary of $VO^{\tau}_{A|B}$ at the point $\left(\mathbf{v}^{\text{opt}}_A - \mathbf{v}^{\text{opt}}_B\right) + \mathbf{u}$. The minimum relative velocity

[1] Available at: https://github.com/aarongarrett/inspyred.
[2] Available at: https://github.com/snape/RVO2.
[3] Available at: https://github.com/sybrenstuvel/Python-RVO2.
[4] Available at: https://github.com/ABojeri/ORCA-EvOp.

change between A and B to avoid any collision within a time τ is then:

$$\mathbf{u} = \left(\underset{\mathbf{v} \in \partial VO^{\tau}_{A|B}}{\arg\min} \left\| \mathbf{v} - \left(\mathbf{v}_A^{\text{opt}} - \mathbf{v}_B^{\text{opt}} \right) \right\| \right) - \left(\mathbf{v}_A^{\text{opt}} - \mathbf{v}_B^{\text{opt}} \right).$$

Symmetrically, the velocity set $ORCA^{\tau}_{B/A}$ is defined for the agent B. For illustration purposes, a graphical representation of $ORCA^{\tau}_{A|B}$ is shown in Fig. 2.

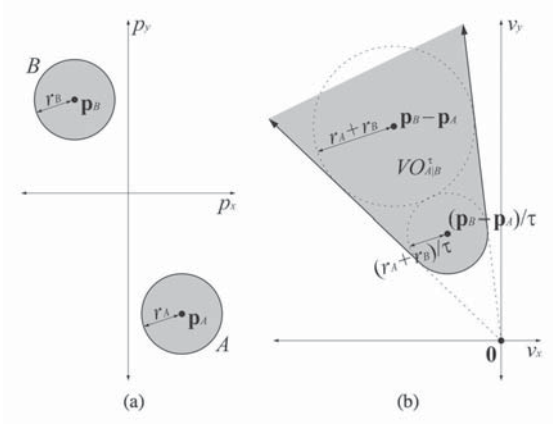

Fig. 1. Graphical representation of Reciprocal Collision Avoidance: in (b) the geometric construction of $VO^{\tau}_{A|B}$ for the agents A and B in (a) [16].

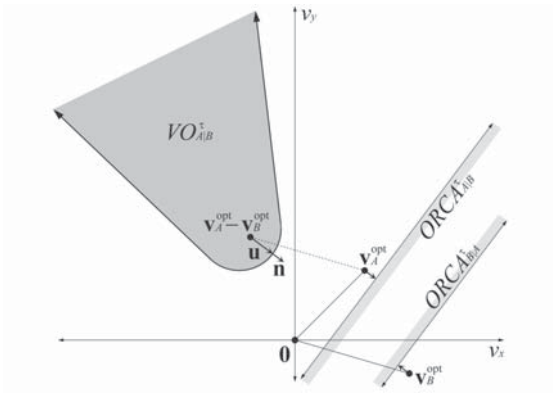

Fig. 2. Graphical representation of $ORCA^{\tau}_{A|B}$ [16].

C. Evolutionary Algorithms

Evolutionary Algorithms (EAs) mimic the process of natural evolution by evolving a "population" of "individuals" (i.e., candidate solutions to a given optimization problem) by means of selection and genetic operators (mutation and/or crossover, also called recombination). As discussed in the introduction, the only information used during the search is the evaluation of the fitness function calculated on the individuals generated along the evolutionary process, i.e. no gradient information is assumed. The overall effect of the selection and genetic operators is to guide the population towards the more promising regions of the search space (i.e., those with better individuals w.r.t. the fitness function), although there is no guarantee to converge to the global optimum if not under specific conditions of the fitness landscape and the population [2]. As an additional note, while the evolutionary metaphor is clear and inspiring it should be remarked that there are some obvious differences between natural evolution and EAs. In particular, in the former fitness is measured *a posteriori* as the reproductive success (number of offspring) of an individual, which in turn depends on how that individual is adapted to the environment. On the contrary, in EAs fitness is measured *a priori* as the evaluation of each individual with respect to the given fitness function, which then affects how many offspring that individual will generate. Another substantial difference is that natural evolution does not foresee any "goal" (it is not an optimization process, although it resembles one) and it is open-ended, while in the case of EAs there is a clear goal (the optimization of the fitness function) and as such they are usually closed-ended, i.e. they are executed until a certain stop criterion is met, e.g. on the number of generations, the execution time or a desired fitness level reached.

III. Implementation

As explained before, in order to optimize the ORCA parameters we used the *RVO2* simulator for testing ORCA, coupled with the *inspyred* library for executing the EA. In this section we will describe the main settings of the two tools.

A. RVO2 Configuration

The *RVO2* library includes a full implementation of the ORCA algorithm, as described in Section II, and allows the simulation of multiple agents navigating in a 2-dimensional space. Furthermore, the library allows the user to have a complete control over several simulation parameters, as well as the properties of the agents and the layout of the obstacles. The main parameters we focused on in our experimentation are listed in Table I. In particular, five of them were considered in the evolutionary optimization process, namely: {*neighborDist*, *maxNeighbors*, *timeHorizon*, *timeHorizonObst*, *maxSpeed*}. These play a fundamental role in performance of the ORCA algorithm, enabling the agents to be more or less efficient in the collision-free navigation. As for the other two parameters, *timeStep* and *radius*, we kept their default values, respectively $1/60$ and 0.1. In fact, these parameters do not depend on the ORCA algorithm itself but rather on the accuracy of the simulator (i.e., the timestep used in its internal numerical integrator) and the size of the agents, which in turn depends on the specific application and available hardware. Therefore, we chose to fix these two parameters instead of taking them into account in the optimization process.

In order to evaluate the optimization of the ORCA parameters under different environment conditions and test the generalization of the parameters across different environments, we devised four different test-cases of increasing complexity in terms of layouts and numbers of obstacles. In each scenario, we considered the presence of four UAVs, that start from the environment's four vertices, whose coordinates are indicated in the second column of Table II. Once the simulation starts, each

TABLE I. RVO2 CONFIGURATION: MAIN SIMULATION PARAMETERS

Parameter	Type (unit)	Meaning
timeStep	float (time)	Timestep of simulation
radius	float (distance)	Maximum proximity (center-to-border)
neighborDist	float (distance)	Maximum proximity (center-to-center) w.r.t. other agents
maxNeighbors	size_t	Maximum number of other agents taken into account for the ORCA algorithm
timeHorizon	float (time)	Minimum amount of time for safe navigation w.r.t. other agents
timeHorizonObst	float (time)	Minimum amount of time for safe navigation w.r.t. obstacles
maxSpeed	float (distance/time)	Maximum speed of agents

TABLE II. RVO2 CONFIGURATION: PROPERTIES OF THE AGENTS

Agent ID	Starting point	Preferred velocity	Target point
0	$(0, 0)$	$(1, 1)$	$(10, 10)$
1	$(10, 0)$	$(-1, 1)$	$(0, 10)$
2	$(10, 10)$	$(-1, -1)$	$(0, 0)$
3	$(0, 10)$	$(1, -1)$	$(10, 0)$

agent is configured with its preferred velocity (third column of Table II), directed towards its target point, that is the opposite vertex w.r.t. its starting point (fourth column of Table II). A graphical representation of the four scenarios is shown in Fig. 3, where the obstacles as well as the borders of the map are depicted in gray and the four circles located at the vertices represent the four agents at their starting positions.

B. Evolutionary Algorithm Configuration

The implementation of the EA was conducted through the use of the *inspyred* library, one of the most adopted frameworks for bio-inspired computation [25]. The library consists of a series of modules that implement all the main features of Evolutionary Algorithms as well as several other bio-inspired algorithms. The modular structure of *inspyred* makes it very easy to apply for solving optimization problems, but also for customizing the optimization algorithms if needed. In *inspyred*, each module is in practice an operator that can be called either at the beginning, during each step, or at the end of the evolutionary loop. In particular, these operators allow among other things to: 1) initialize a population of individuals within user-defined search boundaries (Generator operators); 2) "evolve" these individuals by means of mutation (and/or crossover) and select at every generation the individuals that will "reproduce" (Variator and Selector operators); 3) check if one or more stop criteria are met (Terminator operators); 4) log various aspects of the evolutionary process (Observer operators). One special operator is the Evaluator, which returns the fitness function corresponding to a given individual in input. The complete list of *inspyred* operators, with their description, is reported in Table III.

TABLE III. EA CONFIGURATION: INSPYRED OPERATORS

Operator	Description
Evaluator	Measures the fitness of an individual
Generator	Generates new individuals within the search boundaries
Observer	Stores to file and/or prints the progress of the evolution and other user-defined data
Replacer	Determines the surviving individuals from the previous generation
Selector	Determines the parents for the next generation
Terminator	Determines the evolution termination criteria
Variator	Modifies the input individuals e.g. through mutation and/or crossover

In order to integrate the *RVO2* simulator within *inspyred*, we implemented a custom Observer, a custom Generator, and the necessary Evaluators (one for each scenario). The Observer is designed to save the trajectories corresponding to the best ORCA parameters found by the EA. The Generator creates *pop_size* (this value being the population size) initial individuals, each one consisting of five parameters $\{$*neighborDist*, *maxNeighbors*, *timeHorizon*, *timeHorizonObst*, *maxSpeed*$\}$ uniformly randomly sampled between a lower and an upper bound, defined as in Table IV.

TABLE IV. EA CONFIGURATION: LOWER AND UPPER BOUNDS

Parameter	Lower bound	Upper bound
neighborDist	0.1	5
maxNeighbors	1	3
timeHorizon	0.1	10
timeHorizonObst	0.1	10
maxSpeed	0.5	5

As for the Evaluators, for each scenario we wrapped a call to the *RVO2* simulator that takes as input an individual generated by the EA (so, a set of five ORCA parameters), performs a complete simulation of the movement of the four agents within the respective environment (for a total duration of $100000 \times timeStep$) with those parameters, and finally extracts a series of values (i.e., the number of the collisions, the time to reach the targets, and the distances between each agent and its respective target at the end of the simulation) which are needed for the fitness function calculation to be minimized. The latter was defined to take into account the most significant aspects of the simulation and evaluate the effectiveness of a set of ORCA parameters in terms of accuracy to reach the target and collision avoidance. In particular, the fitness function was formulated as follows:

$$f(\mathbf{x}) = collisions + \overline{error} + \overline{TTR}_{norm} \quad (5)$$

where $\mathbf{x}$ is an individual composed of the five ORCA parameters considered in the optimization process.

The first component of the fitness function, *collisions*, counts the number of collisions occurred between any pair of agents during the simulation. This is computed by updating a counter at each timestep of the simulation. A collision

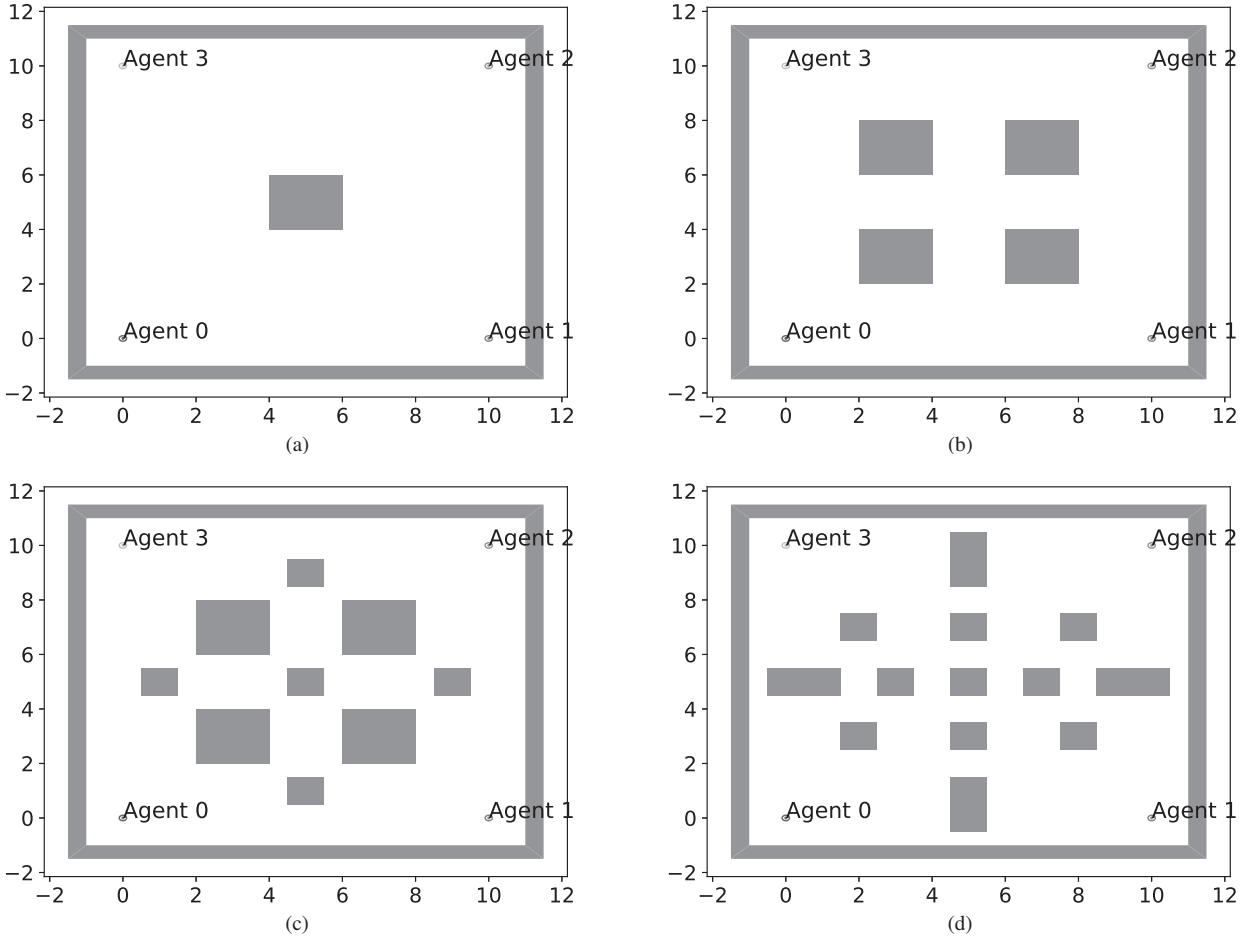

Fig. 3. Graphical representation of the tested scenarios: (a) scenario I, (b) scenario II, (c) scenario III, and (d) scenario IV.

is considered to happen whenever the distance between two agents is smaller than two times their radius (i.e., the *radius* parameter in Table I, set to 0.1), plus a certain safety threshold of 0.01. For clarity, the Python code for the collision check is shown below, where `p1` and `p2` are the 2-dimensional Cartesian coordinates of the two agents, `r` is their radius and `distance()` computes the Cartesian distance between `p1` and `p2`.

```
def check_collisions(p1, p2, r):
  if distance(p1, p2) < (2*r + 0.01):
    return True
  else:
    return False
```

Thus, whenever the `check_collisions()` function above returns `True`, the *collisions* counter is increased.

The second component, $\overline{error}$, represents the distance of each agent from its target position at the end of the simulation, averaged across $n = 4$ agents:

$$\overline{error} = \frac{1}{n}\sum_{i=0}^{n} \text{distance}(\mathbf{p}_i, \mathbf{t}_i) \tag{6}$$

where $\mathbf{p}_i$ is the position of the i-th agent at the end of the simulation, $\mathbf{t}_i$ is the target position, and `distance()` computes the Cartesian distance between the final position and the target.

Finally, to favor simulations where agents are able to reach their target faster, the third component, $\overline{TTR}_{norm}$, computes the time to reach (TTR) the target, again averaged across $n = 4$ agents. The target is considered reached once the agent is within a distance of $radius = 0.1$ from it, in which case the timestep is saved. These timesteps are then averaged across the agents, and the averaged value is normalized in $[0, 1]$ by dividing it by the total number of timesteps of the simulation:

$$\overline{TTR}_{norm} = \frac{\frac{1}{n}\sum_{i=0}^{n} TTR_i}{timeSteps} \tag{7}$$

where TTR_i is the timestep at which the i-th agent reached its target location and $timeSteps$ is the maximum number of timesteps for each simulation (in our experiments, 100000).

It should be noted that due to this fitness function formulation, a hierarchy of relevance is implicitly defined in the optimization phase among the three variables (due to the different orders of magnitude of its three components), in

order to favour first of all the optimization of the number of collisions, then the accuracy to reach the target, and finally the time to reach the target.

To complete the description of the EA, we note that its other modules were all based on existing operators available in *inspyred*, namely:

- *file_observer*: prints to file the statistics of the evolutionary process and the data of each generation;
- *tournament_selection*: performs random sampling of tournaments made of a fixed number of individuals;
- *heuristic_crossover*: performs crossover on two individuals to generate new offspring;
- *gaussian_mutation*: performs Gaussian mutation on one individual to generate new offspring;
- *generational_replacement*: performs generational replacement with elitism;
- *evaluation_termination*: terminates the evolution when the maximum number of evaluations is achieved.

In our experimentation we configured these modules with the parameters shown in Table V.

TABLE V. EA CONFIGURATION: ALGORITHM PARAMETERS

Parameter	Value
pop_size	100
num_selected	*pop_size*
tournament_size	4
mutation_rate	0.4
crossover_rate	0.6
num_elites	1
max_evaluations	100000

IV. RESULTS

For each of the four scenarios, we performed 10 independent runs in order to evaluate the average fitness trend and the ability of the EA to minimize the fitness function independently from the initial population. Fig. 4 shows the average fitness trend $\pm$ std. dev. across the 10 runs performed on each scenario. These trends demonstrate the optimization progress during the evolutionary process. Considering the fitness function formulation in Eq. (5), it should be noted that even if the collisions are equal to 0 and the error to reach the target is 0, the third component will always have a positive value (normalized in $[0, 1]$), due to the finite speed of the agents. In our experiments, as shown in the fitness trends, we have observed that the minimum attainable values are around 0.3. For example, the best set of ORCA parameters found by the EA on scenario IV, reported in Table VI, yield a fitness value of ~ 0.28. A graphical representation showing the agents' trajectories obtained with these parameters is shown in Fig. 5, where it can be seen that the four agents are able to pass through the obstacles and reach their target positions without collisions[5].

[5]We provide some example video recordings of the simulations of the four scenarios at: https://vimeo.com/showcase/7183239.

In order to gain further insight into the optimization results and assess the robustness of EA on problem at hand, we analyzed the distribution of the optimal parameters found in the 10 runs for each scenario. We report in Fig. 6 the histogram of the five ORCA parameters in each scenario. Except for the *timeHorizon* parameter, which converges to different values in the range $[0.1, 10]$ (see Table IV), all the other parameters show a clear trend. In particular, *neighborDist* tends to converge to the largest values, except for scenario I (this is most likely due the the simpler environment of this scenario). *maxNeighbors* converges predominantly (in 39 cases on 40 total runs) to 1, which means that during the computation of the preferred velocities in ORCA each agent takes into account only one of the other three agents. Similarly, we can note that *timeHorizonObst* tends to converge in all runs to the lowest values, while *maxSpeed* tends to the largest values.

Finally, we performed a set of additional tests in order to evaluate the generalization of the results obtained with the EA. In these experiments, we considered the best parameter settings found across the 10 runs for each of the four scenarios, and applied them to the other three scenarios. We then took the mean fitness value across the three scenarios as a measure of generalization. We report the results of these tests in Table VII, where it can be seen that the set of parameters with the best generalization capability is the one optimized for scenario III. This set of parameters, reported in Table VIII, allows to complete successfully every scenario, making each agent reach its target location without colliding with the other agents. By comparing the results in Table VI and Table VIII, we can observe that only *maxNeighbors* and *maxSpeed* take similar values, while the other three parameters are quite different. This means that, in general, it is not possible to guarantee that the parameters optimized for one specific scenario will work in another one, a consideration that should be seriously taken into account in real-world applications.

V. CONCLUSION AND FUTURE WORKS

In this work, we applied an Evolutionary Algorithm to find the optimal parameters of the Optimal Reciprocal Collision Avoidance (ORCA) algorithm. Our solution, based on two existing Python frameworks, namely the *inspyred* library for bio-inspired computation and the *RVO2* library for agent navigation simulations, is general and can be easily customized or adapted to others collision avoidance simulators and/or different optimizers. Our numerical results provide a number of interesting insights as to what concerns the performance of different sets of ORCA parameters in different environment scenarios. Furthermore, we have observed that generalization across scenarios is hard to obtain as the optimal parameters for one scenario are not guaranteed to work in other scenarios.

In future works, it will be interesting to extend this study in different ways. For example, alternative optimization algorithms could be used in comparison with the proposed EA to evaluate if it is possible to obtain better results. In this direction one interesting opportunity would be to evaluate diversity-driven algorithms [26] such as MAP-Elites [27], which com-

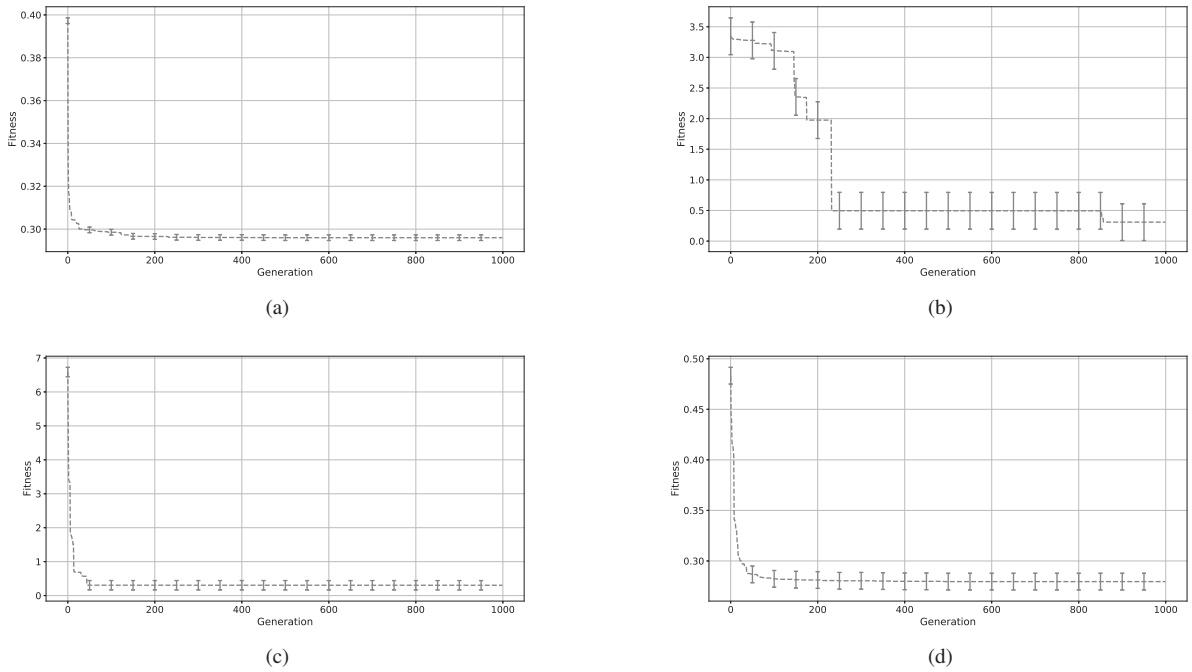

Fig. 4. Fitness trend (average $\pm$ std. dev. across 10 runs) for: (a) scenario I, (b) scenario II, (c) scenario III, and (d) scenario IV.

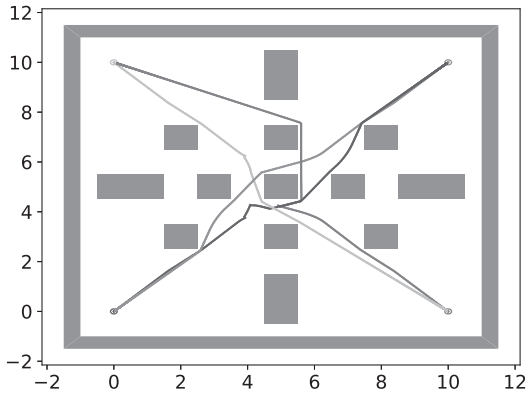

Fig. 5. Graphical representation of the trajectories obtained with the best parameters found for scenario IV (see Table VI).

pared to traditional fitness-driven EAs tend to promote diversity among individuals and as such are capable of better exploration of the parameter space. This kind of algorithms could indeed provide different trade-off optimal parameters corresponding to different (and possibly more generalizable) sets of ORCA parameters. Furthermore, it would also be interesting to test alternative scenarios with a larger number of agents, or groups thereof, or with different (and more complex) environments, in order to evaluate the performances of the optimal ORCA parameters in more difficult conditions.

ACKNOWLEDGMENT

The authors would like to thank Matteo Tadiello for his support in the initial phase of this work.

TABLE VI. BEST PARAMETERS FOUND FOR SCENARIO IV

Parameter	Value
neighborDist	2.2017602006013703
maxNeighbors	1
timeHorizon	2.3891425667157310
timeHorizonObst	0.2388588110888072
maxSpeed	5

TABLE VII. GENERALIZATION RESULTS

	Scenario I	Scenario II	Scenario III	Scenario IV
Mean fitness	4.987	2.653	0.319	4.979

TABLE VIII. BEST GENERALIZATION PARAMETERS (SCENARIO III)

Parameter	Value
neighborDist	4.6351668310830220
maxNeighbors	1
timeHorizon	8.1368729597609600
timeHorizonObst	0.1604237686279340
maxSpeed	4.9999983416259270

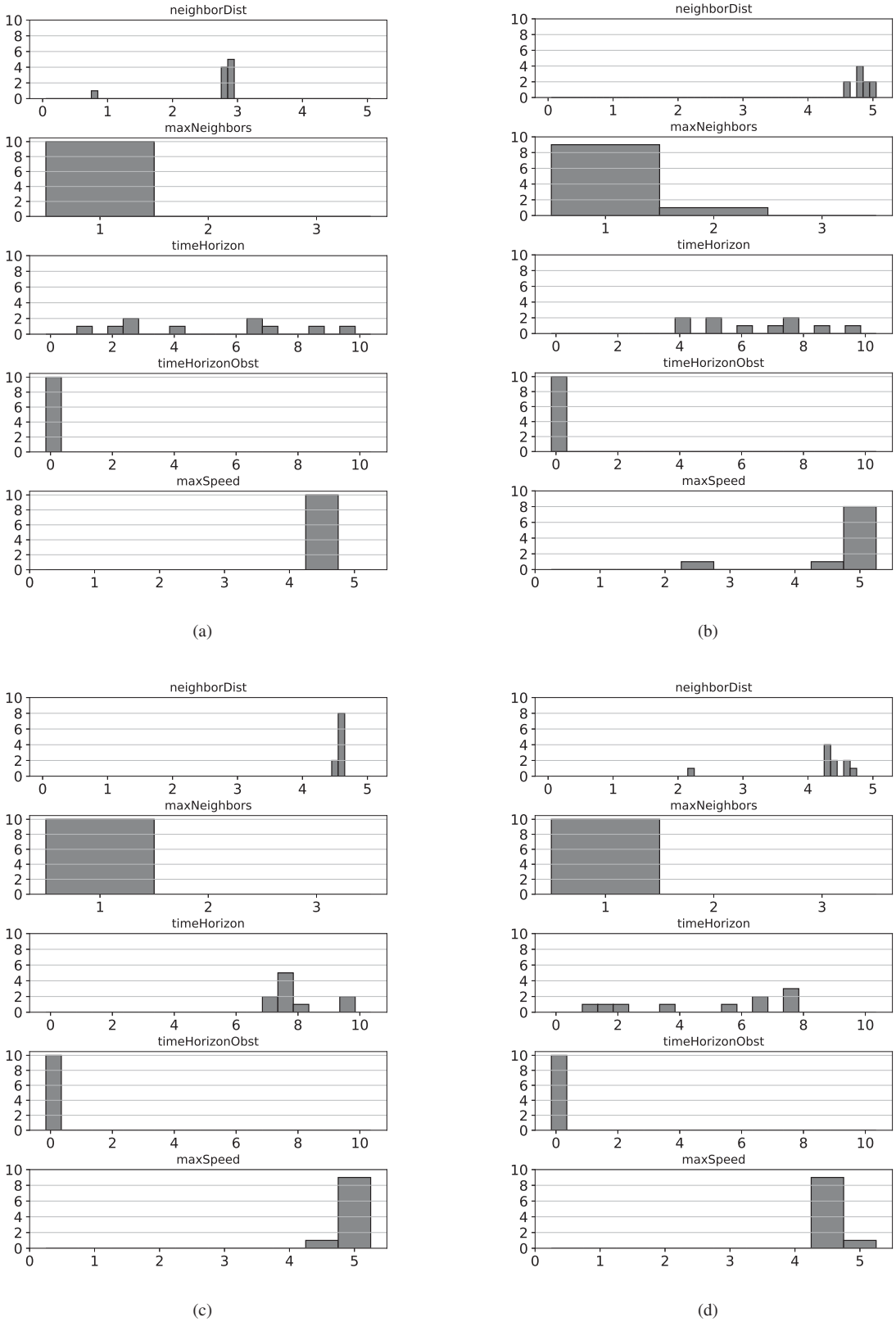

Fig. 6. Parameter distribution (across 10 runs) for: (a) scenario I, (b) scenario II, (c) scenario III, and (d) scenario IV.

REFERENCES

[1] Nikolos, IK and Tsourveloudis, NC and Valavanis, KP, "Evolutionary algorithm based path planning for multiple UAV cooperation," in *Advances in Unmanned Aerial Vehicles*. Springer, 2007, pp. 309–340.

[2] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003.

[3] H. Duan and P. Li, *Bio-inspired computation in unmanned aerial vehicles*. Springer, 2014.

[4] Nikolos, Ioannis K and Valavanis, Kimon P and Tsourveloudis, Nikos C and Kostaras, Anargyros N, "Evolutionary algorithm based offline/online path planner for UAV navigation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 6, pp. 898–912, 2003.

[5] Deb, Kalyanmoy and Pratap, Amrit and Agarwal, Sameer and Meyarivan, TAMT, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[6] Mittal, Shashi and Deb, Kalyanmoy, "Three-dimensional offline path planning for UAVs using multiobjective evolutionary algorithms," in *IEEE Congress on Evolutionary Computation*. IEEE, 2007, pp. 3195–3202.

[7] Pohl, Adam J and Lamont, Gary B, "Multi-objective UAV mission planning using evolutionary computation," in *IEEE Winter Simulation Conference*. IEEE, 2008, pp. 1268–1279.

[8] Roberge, Vincent and Tarbouchi, Mohammed and Labonté, Gilles, "Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning," *IEEE Transactions on industrial informatics*, vol. 9, no. 1, pp. 132–141, 2012.

[9] Lamont, Gary B and Slear, James N and Melendez, Kenneth, "UAV swarm mission planning and routing using multi-objective evolutionary algorithms," in *IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making*. IEEE, 2007, pp. 10–20.

[10] Nikolos, Ioannis K and Brintaki, Athina N, "Coordinated UAV path planning using differential evolution," in *IEEE International Symposium on Mediterranean Conference on Control and Automation Intelligent Control*. IEEE, 2005, pp. 549–556.

[11] Nikolos, Ioannis K and Zografos, Eleftherios S and Brintaki, Athina N, "UAV path planning using evolutionary algorithms," in *Innovations in Intelligent Machines-1*. Springer, 2007, pp. 77–111.

[12] Rathbun, David and Kragelund, Sean and Pongpunwattana, Anawat and Capozzi, Brian, "An evolution based path planning algorithm for autonomous motion of a UAV through uncertain environments," in *IEEE Digital Avionics Systems Conference*, vol. 2. IEEE, 2002, pp. 8D2–8D2.

[13] Duan, Haibin and Luo, Qinan and Shi, Yuhui and Ma, Guanjun, "Hybrid particle swarm optimization and genetic algorithm for multi-UAV formation reconfiguration," *IEEE Computational Intelligence Magazine*, vol. 8, no. 3, pp. 16–27, 2013.

[14] Cekmez, Ugur and Ozsiginan, Mustafa and Sahingoz, Ozgur Koray, "Multi colony ant optimization for UAV path planning with obstacle avoidance," in *IEEE International Conference on Unmanned Aircraft Systems*. IEEE, 2016, pp. 47–52.

[15] Zeng, Yong and Zhang, Rui, "Energy-efficient UAV communication with trajectory optimization," *IEEE Transactions on Wireless Communications*, vol. 16, no. 6, pp. 3747–3760, 2017.

[16] Van den Berg, Jur and Guy, Stephen J. and Lin, Ming and Manocha, Dinesh, "Reciprocal n-Body Collision Avoidance," in *Robotics Research*, Pradalier, Cédric and Siegwart, Roland and Hirzinger, Gerhard, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19.

[17] Van den Berg, Jur and Lin, Ming and Manocha, Dinesh, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 1928–1935.

[18] Snape, Jamie and Van Den Berg, Jur and Guy, Stephen J and Manocha, Dinesh, "Smooth and collision-free navigation for multiple robots under differential-drive constraints," in *IEEE International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 4584–4589.

[19] Alonso-Mora, Javier and Breitenmoser, Andreas and Rufli, Martin and Beardsley, Paul and Siegwart, Roland, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed autonomous robotic systems*. Springer, 2013, pp. 203–216.

[20] Snape, Jamie and Guy, Stephen J and van den Berg, Jur and Lin, Ming C and Manocha, Dinesh, "Reciprocal collision avoidance and multi-agent navigation for video games," in *Workshops at the AAAI Conference on Artificial Intelligence*, 2012.

[21] Alejo, Dominique and Cobano, JA and Heredia, G and Ollero, A, "Optimal reciprocal collision avoidance with mobile and static obstacles for multi-UAV systems," in *IEEE International Conference on Unmanned Aircraft Systems*. IEEE, 2014, pp. 1259–1266.

[22] N. Durand, "Constant speed optimal reciprocal collision avoidance," *Transportation research part C: emerging technologies*, vol. 96, pp. 366–379, 2018.

[23] Arul, Senthil and Sathyamoorthy, Adarsh and Patel, Shivang and Otte, Michael and Xu, Huan and Lin, Ming and Manocha, Dinesh, "LSwarm: Efficient Collision Avoidance for Large Swarms With Coverage Constraints in Complex Urban Scenes," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3940–3947, 2019.

[24] Fiorini, Paolo and Shiller, Zvi, "Motion Planning in Dynamic Environments Using Velocity Obstacles," *The International Journal of Robotics Research*, vol. 17, pp. 760–772, 1998.

[25] Tonda, Alberto, "Inspyred: Bio-inspired algorithms in Python," *Genetic Programming and Evolvable Machines*, pp. 1–4, 2019.

[26] Auerbach, Joshua E and Iacca, Giovanni and Floreano, Dario, "Gaining insight into quality diversity," in *Genetic and Evolutionary Computation Conference Companion*. ACM, 2016, pp. 1061–1064.

[27] Fioravanzo, Stefano and Iacca, Giovanni, "Evaluating MAP-Elites on Constrained Optimization Problems," in *Genetic and Evolutionary Computation Conference Companion*. ACM, 2019, pp. 253–254.