Franklin Open 8 (2024) 100160

Contents lists available at ScienceDirect

Franklin Open

journal homepage: www.elsevier.com/locate/fraope

Extending boids for safety-critical search and rescue

Cole Hengstebeck, Peter Jamieson, Bryan Van Scoy *

Department of Electrical and Computer Engineering, Miami University, 650 East High Street, Oxford, OH, 45056, USA

ARTICLE INFO

MSC:
93A16

Keywords:
Boids algorithm
Search and rescue
Control barrier function

ABSTRACT

Robot swarms can accomplish complex tasks, and in this work, we seek to design swarm robotic algorithms for search and rescue that are scalable to large swarms, efficient in terms of computations, safe from collisions, and tunable to mediate the trade-off between exploration and exploitation in the search. We propose extending the Boids algorithm to accomplish this. Without modifying the three Boids rules of alignment, cohesion, and separation, we add target-seeking and general collision avoidance by using *ghost boids*. Additionally, we use a control barrier function to improve safety at the cost of increased computation. Via simulation in a search and rescue task, we analyze the trade-offs between safety, computational efficiency, and coverage of the environment for our algorithm.

1. Introduction

Swarms of robots may be used to search large and complex environments in search and rescue missions. Traditionally, search and rescue requires teams of people to explore an environment for the target needing help. Scenarios have different requirements, from the number of people searching to the equipment required, such as helicopters or marine vessels. This presents a limitation on the scalability of search tasks, which could, in turn, inhibit the success of a mission.

Robots can replace or supplement people searching and entering potentially dangerous or hard-to-reach areas [1–3]. Robots used for search and rescue vary greatly in design and intended application. For example, drones provide aerial views and cover an area quickly [4], while terrestrial robots have specially designed wheels or treads to maneuver over rubble or up and down stairs [5]. While robots are being used to increase the capabilities of search teams, there is often a one-to-one ratio of robots to human operators.

Several distributed planning and control techniques have been proposed to allow multiple agents to search simultaneously. Particle swarm optimizers have been used to plan a more optimal search path to achieve higher target recovery rates [6,7], and machine learning techniques have been applied to improve path planning for searching [8,9]. Also, control schemes have been proposed that mimic the movement of creatures in biological colonies, which result in successes over some traditional search patterns [10].

Multi-agent search and rescue is a complex task with various design trade-offs. Agents seek to explore their environment while exploiting any available knowledge about the target's location, and they must do so safely and with limited computational abilities. In this work, we aim to design swarm robotic algorithms to safely explore an environment while exploiting information about possible target locations. Such algorithms should have the following properties.

- **Scalable:** The algorithm should scale to large swarms, so robots should only use information from neighbors.
- **Efficient:** The algorithm should be implementable on robots with limited computational abilities.
- **Safe:** Agents should seek to avoid collisions with other agents and environmental obstacles.
- **Tunable:** The algorithm should be tunable to trade-off exploration and exploitation.

One means of achieving verifiable safety in multi-agent systems is using control barrier functions [11–14]. This approach has been successfully applied to various applications such as formation control [15] and can guarantee safety while avoiding deadlocks [16]. However, control barrier functions also have limitations. They guarantee safety when the agents choose their control actions from a certain set, but this set may be empty, implying no safe action exists (e.g., [14]). Moreover, some results are restricted to double-integrator dynamics and rely on braking (subject to a maximum braking force) to avoid collisions (e.g., [12,13]). In contrast, others rely on multiple communication rounds to ensure agreement (e.g., [17]).

Control barrier functions modify a nominal controller in a minimally-invasive manner to improve system safety. As a nominal controller for search and rescue, we consider Boids algorithm [18]. Initially proposed by Craig Reynolds in computer animation to mimic the flocking of birds, this algorithm consists of three simple rules. Each

* Corresponding author.
E-mail address: bvanscoy@miamioh.edu (B. Van Scoy).

https://doi.org/10.1016/j.fraope.2024.100160
Received 5 April 2024; Received in revised form 6 September 2024; Accepted 23 September 2024
Available online 27 September 2024

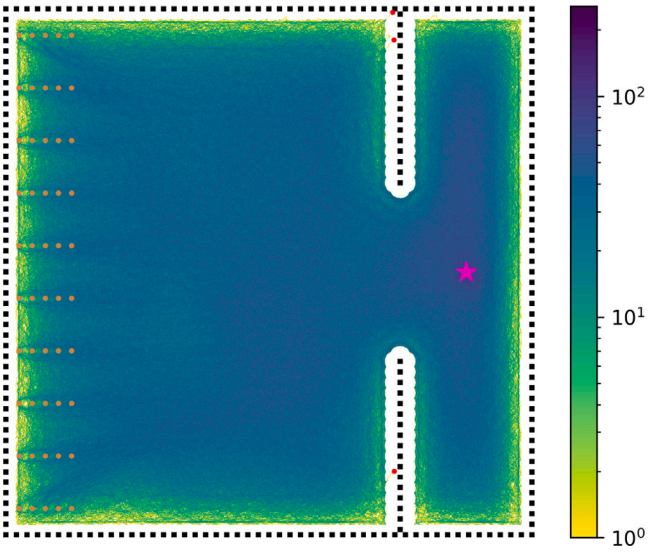


Fig. 1. Heatmap of agent locations aggregated over time for multiple simulations of our algorithm. The agents start on the left (orange) and seek to avoid collisions (red) while navigating around the obstacles (black) to reach the target (magenta) while also exploring the environment. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

agent gathers information about nearby agents, such as their relative position, velocity, and heading, and then decides how to update its movement based on the following concepts: *separation* pushes agents away from each other, *alignment* causes agents to move in the same direction as their neighbors, and *cohesion* clusters agents together to form cohesive flocks.

The Boids algorithm is intentionally simple in design. The algorithm does not provide obstacle avoidance, and there is no means of target-seeking to complete a specific task. Because of this, the Boids algorithm has limited direct application outside of computer animation or agent placement [19,20]. Other algorithms, however, add more rules or complex control on top of Boids to improve it [21,22].

In this paper, we propose a novel extension to the Boids algorithm based on control barrier functions, and we characterize its trade-offs in numerical simulations of a search and rescue task. Without modifying the three rules of separation, alignment, and cohesion, we use *ghost boids* to implement object avoidance and goal-seeking behavior. Ghost boids interact with agents similarly to other agents but with modified rules. One type of ghost boid, an *aid-to-navigation*, is placed on obstacles and the environment boundary to inhibit collisions. Another type of ghost boid, a *compass*, enables the agents to move toward a given location. By controlling the strength of the compass, we can tune how much the algorithm exploits the information to move toward the target location versus how much it explores the environment. To further improve safety, we use a control barrier function to minimally modify the algorithm to avoid collisions at an additional computational cost. Inspired by the Boids algorithm, we impose that each agent travels at a constant velocity so that the environment is continuously being explored, so we cannot guarantee safety via braking. Instead, we show through numerical simulations that safety is vastly improved, even in large swarms.

Fig. 1 illustrates how we present our results. In this figure, a group of 50 agents (shown as orange dots on the left side of the image) explores an environment consisting of a square boundary and two walls (perpendicular lines shown as black squares emerging from the bottom and top of the figure). One agent is given the goal's location (the magenta star), and the agents use local interactions to explore the environment and move toward the goal. To visualize this behavior, we use a heatmap of the agent locations aggregated over time for multiple simulations, where darker regions (dark blue) correspond to well-explored areas and lighter regions (yellow) correspond to the less explored spaces. In this scenario, the agents have effectively explored the entire environment (since much of the heat map is blue). Additionally, three agents are shown to have had collisions across all aggregated simulations, as indicated by the red dots near the walls. By tuning the compass influence, the agents may search closer to the goal location at the cost of less exploration of the environment. The rest of the paper is organized as follows. We describe the problem setup, the simulation parameters, and the Boids algorithm in Section 2. We then introduce the two main components of our algorithm, ghost boids and control barrier functions, in Sections 3 and 4. We describe our results in Section 5, provide a discussion of algorithmic caveats and extensions in Section 6, and concluding remarks in Section 7.

2. Problem setup

To study the properties of our algorithm, we perform various simulations in a search and rescue scenario. For each set of simulation parameters, we run 100 simulations, each for 5000 iterations of the algorithm. Each simulation uses a different random seed for initialization, with agents initialized in a grid on the left side of the environment at random headings, as shown in Fig. 1. Once an agent collides with another agent or an obstacle, it is considered "dead" and becomes stationary.

Our analysis of a swarming multi-agent system (using and extending the Reynolds algorithm) has been done by some other researchers with differing approaches to ours, and their algorithm enhancements have some similarities. Olfati-Saber [23] explores a framework for flocking and includes some similar ideas to ours, which we note later in this work when discussing ghost boids. Ibuki et al. [24] describe a flocking control algorithm that uses control barrier functions that are partially validated regarding safety using a small set of simulations. Beaver et al. [25] provides an overview of the state of flocking algorithms, including Reynolds flocking, which they call Cluster Flocking.

2.1. Simulation parameters

The simulations have numerous parameters that affect the results and make our algorithm tunable. To understand how our algorithm performs in various conditions, we vary the following parameters: number of agents, position of obstacles in the environment, number of informed agents,[1] and influence (or weight) of a compass.

2.2. Performance metrics

Recall that the goal is for the agents to cooperatively search for the target location while maintaining safety. We now describe several performance metrics that we use to characterize how well an algorithm achieves these objectives.

Safety. We characterize the safety of an algorithm by its *survival rate*, which is the ratio of the number of agents that did not have any collisions throughout a given simulation to the total number of agents.

[1] *Informed agents* know the (possible) target location. In our experiments, no knowledge of the target location is spread; only agents that are initialized as informed ever know the target location.

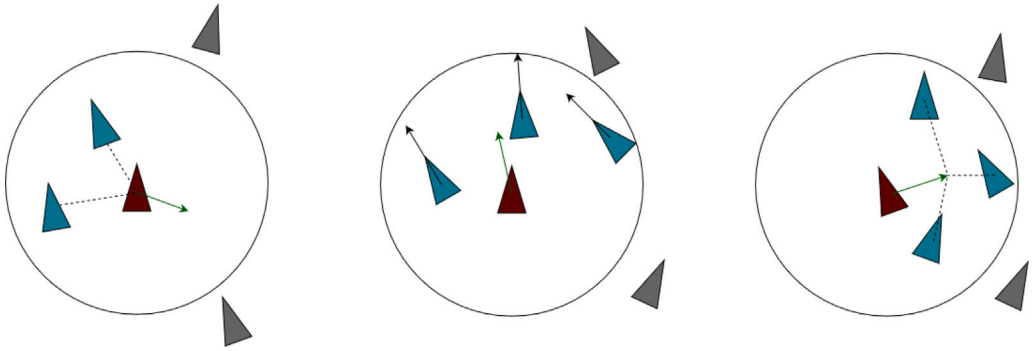


Fig. 2. Visualization of Boids rules. Left to right: separation, alignment, and cohesion.

Performance. We have two metrics to characterize how well a group of agents explores an environment while exploiting information about the possible target location. Our first metric is the *target success rate*, which is the ratio of agents that have reached the target location by the end of the simulation to the total number of agents. We count an agent as having reached the target if it collides with the target at any point during the simulation, even if the agent later moves away from the target and/or has a separate collision and dies. While the target success rate indicates how well the agents exploit the information about the possible target location, it does not characterize how well they explore the remainder of the environment. As it is difficult to capture this exploration/exploitation trade-off in a single number, we instead use a *heatmap* of agent positions aggregated over both iterations and simulations to study the search coverage of an algorithm. Dark regions of the heatmap indicate locations searched many times over the simulations (possibly by multiple agents), while lighter regions indicate low search coverage. An example heatmap is shown in Fig. 1.

2.3. The boids algorithm

We now describe the traditional Boids algorithm that our algorithm builds upon. In this algorithm, each agent moves in the two-dimensional plane with unit speed in the direction of its heading. The heading is a weighted average of the headings obtained from the three rules of separation, alignment, and cohesion. For each agent, the heading from the separation rule points in the opposite direction as the relative positions of its neighbors, the heading from the alignment rule points in the average direction as the headings of its neighbors, and the heading from the cohesion rule points in the direction of the average relative positions of its neighbors. We illustrate these three rules in Fig. 2.

To motivate the need for obstacle avoidance and goal-seeking behavior, consider running the Boids algorithm in a bounded environment with no obstacles. The corresponding heatmap is shown in Fig. 3, where agents wander through the environment until colliding with the boundary (collisions shown in red). In the following two sections, we describe our extensions to this algorithm that address these issues.

3. Ghost boids

We first extend the traditional Boids algorithm to have object avoidance and goal-seeking behavior through the use of *ghost boids*. These are identical to normal agents, but they do not actively move on their own. Ghost boids affect the rules for separation and alignment for neighboring agents. There are two types of ghost boids: aids to navigation (ATON) [26] and compasses, which we now discuss.

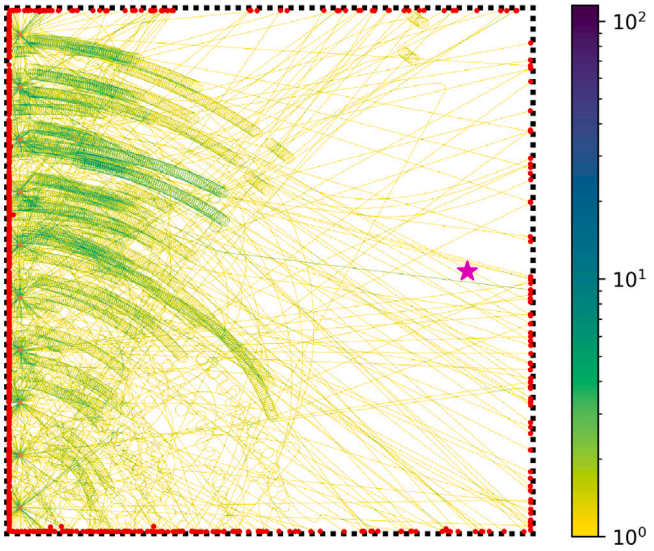


Fig. 3. Heatmap of the traditional Boids algorithm, which has no obstacle-avoidance or goal-seeking behavior.

3.1. Aids to Navigation (ATON)

We use ATONs to provide obstacle avoidance. ATONs are placed around the perimeter of obstacles and the environment, as shown in Fig. 4 (red). These ghost boids are aligned such that they face away from danger, so they point away from the center of an obstacle and inwards from the boundary of the environment.

ATONs affect the separation and alignment rules of neighboring agents. When an agent approaches danger, the ATON repels it from its location while also aligning the agent's heading away from danger.

Our concept of ghost boids is not the first approach to obstacle avoidance. Olfati-Saber [23] proposed what they call γ-agents that direct a boid away from obstacles. The difference between this approach and ours is that our ATONs are fixed entities that can be physically realized — this is both a pro and con. The γ-agents must be virtually created within the algorithm and placed at the closest point on the obstacle from the physical agent. In contrast, ATONs represent fixed entities, such as AprilTags [27,28], that must be placed in the environment and can be detected by the agent.

3.2. Compasses

Compasses are another type of ghost boid that adds goal-seeking capabilities to the algorithm. A set of agents called *informed agents* know a possible target location to explore. For instance, agents may be informed through initialization or the spread of information from

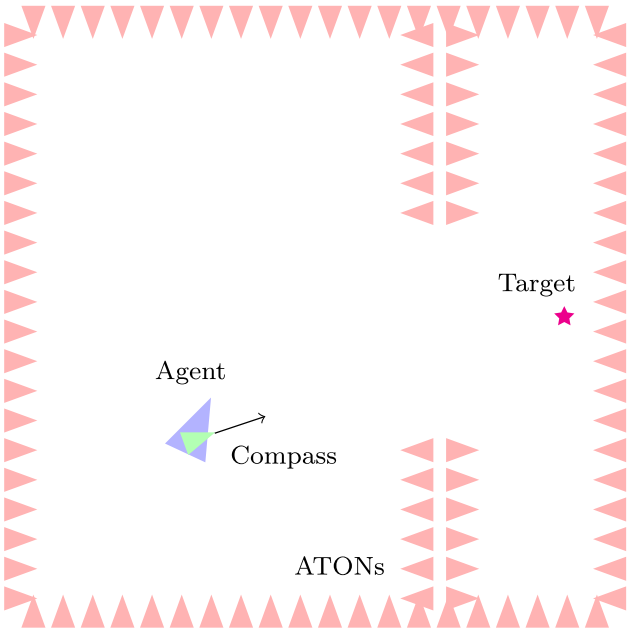


Fig. 4. Illustration of ghost boids. ATONs (red triangles) point away from danger for collision avoidance. A compass (green) is located on an informed agent (blue) and points toward the target (magenta) for goal-seeking behavior. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

neighboring agents that are informed. Olfati-Saber [23] first proposed a similar concept with their β-agents that provide the algorithm with per agent direction.

Each compass is assigned to a single agent and maintains the same position as the agent. The compass only affects the alignment rule of the agent to which it is assigned, and its heading points directly toward the target location, as illustrated in Fig. 4 (green). To avoid the influence of the compass being attenuated by several neighbors, we scale the compass's influence (or weight) proportional to the number of neighboring agents and ATONs (but not other compasses).

4. Control barrier functions

While the separation rule of the Boids algorithm and the ATON ghost boids provide some level of collision avoidance, they provide insufficient safety in scenarios with large clusters of agents. For instance, consider a set of 50 agents, none of which are informed, in a bounded environment with two walls. The corresponding heatmap is shown in Fig. 5, with many collisions occurring both at obstacles and in the interior of the environment between agents.

To further improve safety, we use a constraint-based control methodology inspired by long-duration autonomy in ecology [29]. In particular, we use a control barrier function (CBF) to modify the algorithm in a computationally efficient way that promotes safety [11]. Let the two-dimensional vectors p_i and v_i denote the position and velocity of boid i (either a physical agent or a ghost boid). Let $s_i = (p_i, v_i)$ denote the state of boid i, and let $s = \{s_i\}$ denote the aggregated state of all boids.

To characterize safety, we let S denote the set of aggregated states that are considered safe. For instance, a state may be safe if no agent collides with an obstacle or any other agent, j. Suppose we can describe the safe set as the super-level set of some barrier function h applied to all pairs of boids,

$$S = \{s \mid h(s_i, s_j) \geq 0 \text{ for all } i, j\}. \tag{1}$$

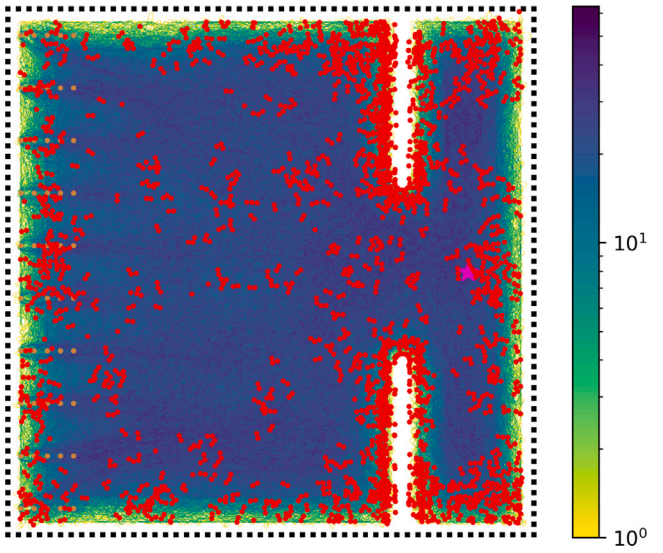


Fig. 5. Heatmap of the Boids algorithm with ghost boids. Many collisions occur (red) even with the separation rule and ATONs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

For collision avoidance, we choose the barrier function such that it is nonnegative when two boids are separated by at least some positive distance D,

$$h(s_i, s_j) = \|p_i - p_j\|^2 - D^2. \tag{2}$$

To increase safety, we set the safety distance larger than the distance at which two agents would collide.

The idea behind CBFs is to choose the heading so that the barrier function does not become too small and the state remains safe. In continuous time, this safety constraint takes the form

$$\frac{\mathrm{d}}{\mathrm{d}t} h(s_i, s_j) + \alpha\, h(s_i, s_j) \geq 0 \quad \text{for all } i, j \tag{3a}$$

for some positive constant α, where $\frac{\mathrm{d}}{\mathrm{d}t} h$ is the derivative of the barrier function for time along the system's dynamics. Alternatively, in discrete time, the safety constraint is [30]

$$h(s_i^+, s_j^+) \geq c\, h(s_i, s_j) \quad \text{for all } i, j \tag{3b}$$

for some constant $c \in (0, 1)$, where s_i^+ denotes the state of boid i at the next iteration of the algorithm.

Denote the velocity of boid i using the Boids algorithm with ghost boids as v_i^{nom}. The CBF controller modifies this velocity in a minimally invasive way while ensuring safety. It chooses the velocity of each agent to minimize the squared norm of the difference between its velocity and that of the nominal velocity subject to the safety constraint:

$$\begin{aligned} v_i \quad = \quad & \arg\min_{v} \quad \|v - v_i^{\text{nom}}\|^2 \\ & \text{subject to safety constraint (3a) or (3b).} \end{aligned} \tag{4}$$

If the nominal velocity v_i^{nom} satisfies the safety constraint, it is trivially the optimal solution. If not, the optimizer finds the velocity as close to the nominal velocity as possible without defying the safety constraint.

We now describe the detailed formulation of the optimization problem (4) for the barrier function (2). We first consider the safety constraint in discrete time. Using a forward discretization of the differential equation $\frac{\mathrm{d}}{\mathrm{d}t} p_i = v_i$ with time step Δt, the position of boid i at the next iteration is $p_i^+ = p_i + \Delta t\, v_i$. The safety constraint (3b) between boids i and j is then

$$\begin{bmatrix} p_i - p_j \\ v_i - v_j \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} 1 - c & \Delta t \\ \Delta t & \Delta t^2 \end{bmatrix} \begin{bmatrix} p_i - p_j \\ v_i - v_j \end{bmatrix} \geq (1 - c)\, D^2,$$

which is quadratic in the differences between the positions and velocities of the two boids. In this formulation, the optimization problem (4) is a quadratically-constrained quadratic program, which is, in general, NP-hard to solve. Furthermore, this is a global optimization problem since it couples the velocities between all of the agents, which are the variables in the problem.

Since the previous formulation is not computationally tractable, we propose a modified problem that is easier to solve but does not guarantee safety. Define the positive constant $\alpha = \Delta t^{-1}(1-c)$. For Δt sufficiently small, we can neglect the term in the above inequality that is quadratic in the relative velocities to obtain the modified safety constraint

$$\begin{bmatrix} p_i - p_j \\ v_i - v_j \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \alpha & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p_i - p_j \\ v_i - v_j \end{bmatrix} \geq \alpha D^2. \tag{5}$$

This constraint is *affine* in the velocity. The optimization problem is then a convex quadratic program that can be solved efficiently using standard numerical solvers [31–34]. In addition, this optimization problem can be solved in a decentralized way by having each agent i implement half of the constraint:

$$(p_i - p_j)^{\mathsf{T}} v_i + \tfrac{\alpha}{4}(\|p_i - p_j\|^2 - D^2) \geq 0, \tag{6}$$

which does not depend on the velocity of boid j (see [29]). Therefore, the optimization problem (4) for each boid only requires the relative positions and velocities of neighboring boids.

To further motivate this simplification, we observe that the modified safety constraint (5) is precisely the safety constraint (3a) for the *continuous-time* dynamics. Since we implement the system in discrete-time, this continuous-time safety constraint is not sufficient to guarantee safety. However, our simulations show it drastically improves safety at a moderate additional computational cost.

Beyond the simplification of the safety constraint, another possible source of collisions is that instead of applying the optimal solution v_i from (4), we apply the *normalized*[2] velocity $\|v_i\|^{-1} v_i$. We do not include this normalization as a constraint in (4) as that would again lead to a quadratically-constrained quadratic program. Without requiring a normalized velocity, we could achieve guaranteed safety simply by having all agents remain stationary (e.g., see the braking mechanism in [13]).

To summarize, each agent i first finds its nominal velocity v_i^{nom} by applying the Boids algorithm with ghost boids, solves the quadratic program (4) subject to its portion of the modified safety constraint in (6) for all neighboring agents and ATONs to obtain the velocity v_i, and then applies the normalized velocity $\|v_i\|^{-1} v_i$ to update its position.

5. Experimental results

We now illustrate the properties of our algorithm using simulations of the search and rescue scenario described in Section 2 under various parameter settings. Since it is not realistic to discuss all possible combinations of parameters, we focus on the parameters most relevant to the particular discussion.

5.1. Safety vs. Efficiency

In physical systems, hardware constraints limit the computational complexity of the algorithms they can implement. Therefore, we first study the trade-off between safety and computational efficiency. Fig. 6 shows the heatmap produced by 10 agents, none of which are informed, in an open environment.

[2] We could directly apply the velocity obtained from the optimization problem (4). The Boids algorithm, however, traditionally uses a constant velocity, which is why we choose to use the normalized velocity.

Table 1
Survival rate in percent (average runtime in milliseconds) of each algorithm with no informed agents in an open environment with a varying number of agents.

Algorithm	5 agents		10 agents		50 agents		100 agents	
Traditional	1.6	(40)	2.4	(70)	0.0	(326)	0.0	(667)
Ghost boids	100	(77)	95.4	(137)	46.2	(593)	23.9	(1086)
CBF	100	(453)	100	(1360)	99.9	(11 352)	99.9	(25 528)

The most computationally efficient algorithm is the traditional Boids algorithm, shown in Fig. 6(a), as it only requires each agent to compute weighted sums of the relative positions and velocities of its neighboring agents. Since this algorithm has no obstacle avoidance mechanism, agents wander through the environment until colliding with the boundary.

The ATONs placed around the boundary result in much fewer collisions, as illustrated in Fig. 6(b). Most agents wander through the environment throughout the simulation, resulting in a darker heatmap. However, the repulsive effect of the separation rule is not always strong enough to keep agents from colliding, resulting in collisions between agents throughout the environment. This algorithm has a slightly higher computational cost as the weighted sums now include the ATONs on the boundary.

To further improve safety, the CBF algorithm from Section 4 enforces the safety constraint to be satisfied if such a velocity exists. This results in no collisions in this scenario, although it requires each agent to solve a quadratic program at each iteration of the algorithm. This optimization problem is trivial, however, if the nominal velocity already satisfies the safety constraint; it only incurs an additional cost when the nominal velocity is unsafe.

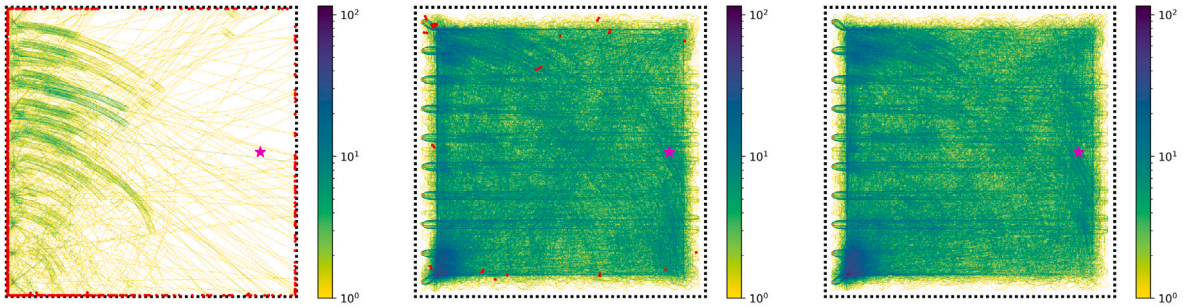
The survival rate and runtime for each algorithm in this scenario are provided in Table 1 for various agents. The traditional Boids algorithm is the most computationally efficient but the least safe, while the CBF algorithm is the most safe but the least efficient. All algorithms become less safe and have longer runtimes as the number of agents increases.

While Table 1 shows the average *total* runtime, the more relevant statistic when considering viability of real-time applications is the *per agent* runtime. We can estimate this quantity by dividing the total runtime by the number of agents. This is not an exact measure of the runtime per agent as it also includes the initialization time, so we expect the runtime per agent to decrease as the number of agents increases since the initialization time gets distributed across a larger number of agents. This holds true for the Traditional and Ghost boid algorithms. The per-agent runtime of the CBF algorithm, however, increases as the number of agents increases. This is due to the complexity of solving the quadratic programs being such a large factor in the CBF algorithm. As the number of agents increases, the average number of neighbors of each agent also increases. This increases the number of constraints in the quadratic program, making solving it more computationally expensive.

5.2. Exploration vs. Exploitation

We now study the trade-off between exploring the environment and exploiting information about the possible target location available to informed agents. Two parameters affect this trade-off: the number of informed agents and the compass influence.

The compass influence is a single scalar parameter that trades off exploration and exploitation. Consider a group of 50 agents, all of which are informed, running the CBF algorithm in an environment with a wall as shown in Fig. 7 for varying amounts of compass influence. Without the compass (Fig. 7(a)), the agents uniformly cover the environment with few collisions due to the CBF mechanism. With a small amount of compass influence (Fig. 7(b)), the agents still explore the environment, but to a lesser extent as they spend more time near the target location due to the compass pointing in that direction. With

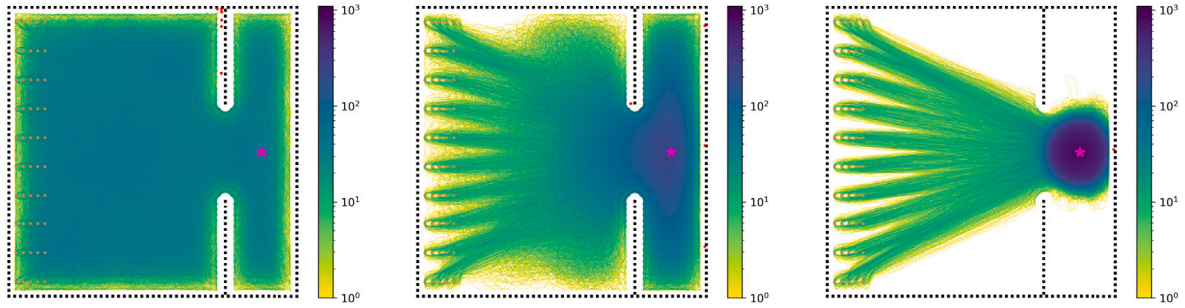


(a) The traditional boids algorithm has many collisions at the boundary.

(b) ATONs repel agents from the boundary causing fewer collisions.

(c) The CBF algorithm has no collisions, but agents solve quadratic programs at each iteration.

Fig. 6. Heatmaps for each algorithm. The system can be made more safe at additional computational cost.

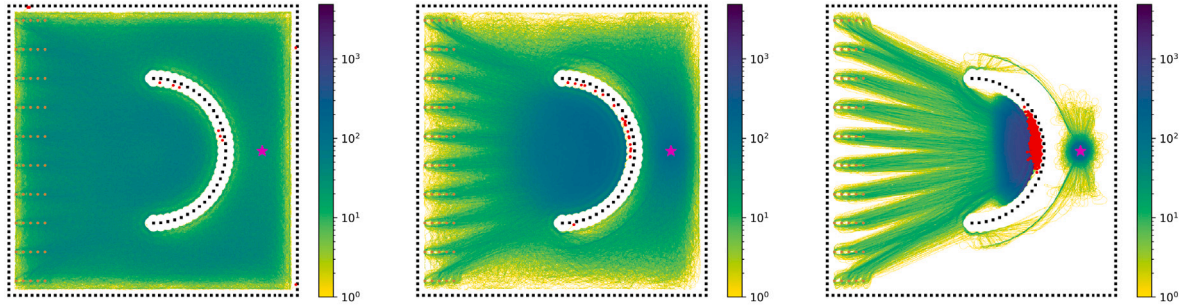


(a) 0% compass influence

(b) 10% compass influence

(c) 100% compass influence

Fig. 7. Heatmaps of the CBF algorithm. A larger compass influence yields more exploitation and less exploration.



(a) 0% compass influence

(b) 10% compass influence

(c) 100% compass influence

Fig. 8. Heatmaps of the CBF algorithm. Too much compass influence results in agents being trapped in the bowl.

100% compass influence (Fig. 7(c)), the agents move directly toward the target location and remain within a ball about the target.

Similar to the compass influence, the number of informed agents can also be used to exploit the trade-off between exploration and exploitation. More informed agents result in more exploitation, similar to using more compass influence. As the results are qualitatively similar, we do not show the plots due to space limitations.

While more compass influence results in more exploitation of target location information, it can also result in agents becoming trapped by environmental obstacles. Consider the same setup as before but in an environment with a bowl-shaped obstacle, as illustrated in Fig. 8. Here, the target is located on the side of the bowl opposite the starting location of the agents. When the compass influence is too large, many agents become trapped by the bowl and are, therefore, unable to reach the target location. In addition, this results in many collisions due to the high density of agents inside the bowl. We conclude that complex environments require a smaller compass influence (or fewer informed agents) so that agents explore enough of the environment to locate the target.

6. Discussion

Our algorithm could be extended in various ways. We made several simplifying assumptions in deriving the safety constraint for the CBF, so our algorithm does not guarantee safety. An interesting problem is constructing a safety constraint for the discrete-time dynamics that is always feasible to guarantee safety.

Another extension is to design a high-level planner on top of our algorithm to prevent agents from becoming stuck by obstacles. In this work, we only considered cases where all informed agents have access to the same possible target location. In a more realistic search and rescue scenario, there may be many possible locations where the target may be located. The algorithm should then trade off the amount of effort spent searching each possible target location with the rest of the environment. Beyond a finite number of single points, we may generally have a probability distribution over the environment that describes the probability that the target is at any given location. In that case, we would want to spend time in each region proportional to the probability that the target is in that region.

Finally, implementing these algorithms on a physical swarm of robots would provide valuable insight into their viability in realistic search and rescue applications. While simulations provide general insights, imperfect sensor readings and actuator control, dynamic terrain, different robot specifications, and more will result in different performances in realistic applications.

7. Conclusions

In this work, we extended the traditional Boids algorithm to be applicable to search and rescue tasks. We introduced two types of ghost boids, ATONs, and compasses, to give the algorithm obstacle-avoidance and goal-seeking behavior. Then, we implemented a CBF controller on top of this algorithm to achieve better safety at an additional computational cost. We showed through simulations that our algorithm can trade off safety and efficiency through ATONs and CBFs and that the number of informed agents and compass influence can be used to trade off exploration of the environment with the exploitation of possible target location information. However, in complex environments (such as the bowl-shaped obstacle), too much compass influence can cause livelock scenarios in which informed agents cannot navigate around the obstacle to reach the target.

CRediT authorship contribution statement

Cole Hengstebeck: Writing – original draft, Visualization, Validation, Software, Formal analysis. **Peter Jamieson:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Bryan Van Scoy:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research received no external funding.

References

[1] S. Waharte, N. Trigoni, Supporting search and rescue operations with UAVs, in: International Conference on Emerging Security Technologies, 2010, pp. 142–147, http://dx.doi.org/10.1109/EST.2010.31.

[2] J.L. Baxter, E. Burke, J.M. Garibaldi, M. Norman, Multi-Robot Search and Rescue: A Potential Field Based Approach, Springer, Berlin, Heidelberg, 2007, pp. 9–16, http://dx.doi.org/10.1007/978-3-540-73424-6_2.

[3] J. León, G.A. Cardona, A. Botello, J.M. Calderón, Robot swarms theory applicable to seek and rescue operation, in: Intelligent Systems Design and Applications: 16th International Conference on Intelligent Systems Design and Applications, Springer, Porto, Portugal, 2017, pp. 1061–1070.

[4] A. Matos, A. Martins, A. Dias, B. Ferreira, J.M. Almeida, H. Ferreira, G. Amaral, A. Figueiredo, R. Almeida, F. Silva, Multiple robot operations for maritime search and rescue in euRathlon 2015 competition, in: OCEANS, Shanghai, China, 2016, pp. 1–7, http://dx.doi.org/10.1109/OCEANSAP.2016.7485707.

[5] R.R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, A.M. Erkmen, Search and Rescue Robotics, Springer, Berlin, Heidelberg, 2008, pp. 1151–1173, http://dx.doi.org/10.1007/978-3-540-30301-5_51.

[6] K.J. Rafferty, E.W. McGookin, An autonomous air-sea rescue system using particle swarm optimization, in: International Conference on Connected Vehicles and Expo, 2013, pp. 459–464, http://dx.doi.org/10.1109/ICCVE.2013.6799836.

[7] Z. Chen, H. Liu, Y. Tian, R. Wang, P. Xiong, G. Wu, A particle swarm optimization algorithm based on time-space weight for helicopter maritime search and rescue decision-making, IEEE Access 8 (2020) 81526–81541, http://dx.doi.org/10.1109/ACCESS.2020.2990927.

[8] P. Xiong, L. Hu, T. Yongliang, C. Zikun, W. Bin, Y. Hao, Helicopter maritime search area planning based on a minimum bounding rectangle and K-means clustering, Chin. J. Aeronaut. 34 (2) (2021) 554–562, http://dx.doi.org/10.1016/j.cja.2020.08.047.

[9] B. Ai, M. Jia, H. Xu, J. Xu, Z. Wen, B. Li, D. Zhang, Coverage path planning for maritime search and rescue using reinforcement learning, Ocean Eng. 241 (2021) 110098, http://dx.doi.org/10.1016/j.oceaneng.2021.110098.

[10] W. Yue, Y. Xi, X. Guan, A new searching approach using improved multi-ant colony scheme for multi-UAVs in unknown environments, IEEE Access 7 (2019) 161094–161102, http://dx.doi.org/10.1109/ACCESS.2019.2949249.

[11] A.D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, P. Tabuada, Control barrier functions: Theory and applications, in: 18th European Control Conference, 2019, pp. 3420–3431, http://dx.doi.org/10.23919/ECC.2019.8796030.

[12] U. Borrmann, L. Wang, A.D. Ames, M. Egerstedt, Control barrier certificates for safe swarm behavior, IFAC-PapersOnLine 48 (27) (2015) 68–73, http://dx.doi.org/10.1016/j.ifacol.2015.11.154.

[13] L. Wang, A.D. Ames, M. Egerstedt, Safety barrier certificates for collisions-free multirobot systems, IEEE Trans. Robot. 33 (3) (2017) 661–674, http://dx.doi.org/10.1109/TRO.2017.2659727.

[14] M. Machida, M. Ichien, Consensus-based control barrier function for swarm, in: 2021 IEEE International Conference on Robotics and Automation, ICRA, 2021, pp. 8623–8628, http://dx.doi.org/10.1109/ICRA48506.2021.9561971.

[15] B.A. Butler, C.H. Leung, P.E. Paré, Collaborative safe formation control for coupled multi-agent systems, in: European Control Conference, ECC, 2024, pp. 3410–3415, http://dx.doi.org/10.23919/ECC64448.2024.10590880.

[16] M. Jankovic, M. Santillo, Collision avoidance and liveness of multi-agent systems with CBF-based controllers, in: 60th IEEE Conference on Decision and Control, CDC, 2021, pp. 6822–6828, http://dx.doi.org/10.1109/CDC45484.2021.9682854.

[17] B.A. Butler, P.E. Paré, Collaborative safety-critical control for dynamically coupled networked systems, 2024, arXiv:2310.03289, URL https://arxiv.org/abs/2310.03289.

[18] C.W. Reynolds, Flocks, herds and schools: A distributed behavioral model, SIGGRAPH Comput. Graph. 21 (4) (1987) 25–34, http://dx.doi.org/10.1145/37402.37406.

[19] P.M. Mavhemwa, I. Nyangani, Uniform spatial subdivision to improve Boids Algorithm in a gaming environment, Int. J. Adv. Res. Dev. 3 (10) (2018) 49–57.

[20] J. Hagelbäck, Hybrid pathfinding in StarCraft, IEEE Trans. Comput. Intell. AI Games 8 (4) (2016) 319–324, http://dx.doi.org/10.1109/TCIAIG.2015.2414447.

[21] J.B. Clark, D.R. Jacques, Flight test results for UAVs using boid guidance algorithms, Procedia Comput. Sci. 8 (2012) 232–238, http://dx.doi.org/10.1016/j.procs.2012.01.048.

[22] J. Wang, H. Zhao, Y. Bi, S. Shao, Q. Liu, X. Chen, R. Zeng, Y. Wang, L. Ha, An improved fast flocking algorithm with obstacle avoidance for multiagent dynamic systems, J. Appl. Math. 2014 (1) (2014) 659805, http://dx.doi.org/10.1155/2014/659805.

[23] R. Olfati-Saber, Flocking for multi-agent dynamic systems: Algorithms and theory, IEEE Trans. Autom. Control 51 (3) (2006) 401–420, http://dx.doi.org/10.1109/TAC.2005.864190.

[24] T. Ibuki, S. Wilson, J. Yamauchi, M. Fujita, M. Egerstedt, Optimization-based distributed flocking control for multiple rigid bodies, IEEE Robot. Autom. Lett. 5 (2) (2020) 1891–1898, http://dx.doi.org/10.1109/LRA.2020.2969950.

[25] L.E. Beaver, A.A. Malikopoulos, An overview on optimal flocking, Annu. Rev. Control 51 (2021) 88–99, http://dx.doi.org/10.1016/j.arcontrol.2021.03.004.

[26] United States Coast Guard, 13th District, Office of Boating Safety, U.S. aids to navigation system, 2018, URL https://www.pacificarea.uscg.mil/Portals/8/District_13/dpw/docs/usaidstonavigationbooklet.pdf?ver=2018-10-15-154501-363.

[27] E. Olson, AprilTag: A robust and flexible visual fiducial system, in: IEEE International Conference on Robotics and Automation, 2011, pp. 3400–3407, http://dx.doi.org/10.1109/ICRA.2011.5979561.

[28] J. Wang, E. Olson, AprilTag 2: Efficient and robust fiducial detection, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2016, pp. 4193–4198, http://dx.doi.org/10.1109/IROS.2016.7759617.

[29] M. Egerstedt, Robot Ecology: Constraint-Based Design for Long-Duration Autonomy, Princeton University Press, Princeton, 2021, http://dx.doi.org/10.1515/9780691230078.

[30] A. Agrawal, K. Sreenath, Discrete control barrier functions for safety-critical control of discrete systems with application to bipedal robot navigation, in: Robotics: Science and Systems, Vol. 13, Cambridge, MA, USA, 2017, URL https://www.roboticsproceedings.org/rss13/p73.pdf.

[31] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, S. Boyd, OSQP: an operator splitting solver for quadratic programs, Math. Program. Comput. 12 (4) (2020) 637–672, http://dx.doi.org/10.1007/s12532-020-00179-2.

[32] G. Banjac, P. Goulart, B. Stellato, S. Boyd, Infeasibility detection in the alternating direction method of multipliers for convex optimization, J. Optim. Theory Appl. 183 (2) (2019) 490–519, http://dx.doi.org/10.1007/s10957-019-01575-y.

[33] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, S. Boyd, Embedded code generation using the OSQP solver, in: IEEE Conference on Decision and Control, 2017, http://dx.doi.org/10.1109/CDC.2017.8263928.

[34] B. Stellato, V.V. Naik, A. Bemporad, P. Goulart, S. Boyd, Embedded mixed-integer quadratic optimization using the OSQP solver, in: European Control Conference, 2018, http://dx.doi.org/10.23919/ECC.2018.8550136.