Extending Boids for Safety-Critical Search and Rescue

Cole Hengstebeck Peter Jamieson Bryan Van Scoy

***Abstract*— Robot swarms can be used to accomplish complex tasks that are difficult or impossible for any single agent alone. In this paper, 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 of the environment and exploitation of possible target location information. To accomplish this, we propose extending the classical Boids algorithm from computer animation. Without modifying the three Boids rules of alignment, cohesion, and separation, we add target-seeking and general collision avoidance through the novel use of *ghost boids*. In addition, we use a control barrier function to further improve safety at the cost of increased computations. Through simulations in a search and rescue task, we analyze the trade-offs between safety, computational efficiency, and coverage of the environment for our algorithm.**

I. 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 in need of 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 are used to 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 the use of 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].

One particularly simple swarm robotics algorithm is the Boids algorithm [11]. Originally proposed by Craig Reynolds in computer vision to mimic the flocking of birds, this algorithm consists of three simple rules to mimic the flocking behavior of birds. Each agent gathers information about nearby agents, such as their relative position, velocity, and heading, and decides how to update its movement based on this information according to three rules: *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.

All authors are with the Department of Electrical and Computer Engineering, Miami University, Oxford, OH 45056, USA.
Emails: colehengstebeck@gmail.com, jamiespa@miamioh.edu, bvanscoy@miamioh.edu

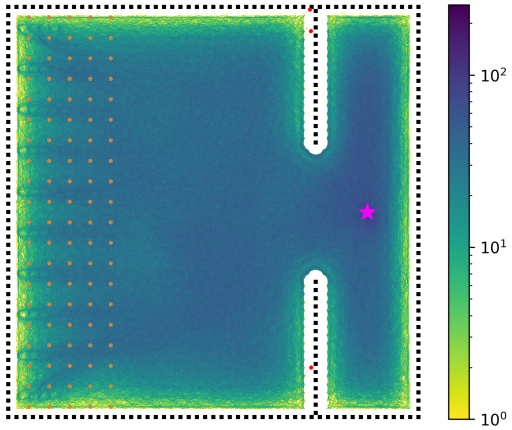


Fig. 1: Heatmap of agent locations aggregated over time for multiple simulations of our algorithm. The agents start on the left (brown) and seek to avoid collisions (red) while navigating around the obstacles (black) to reach the target (magenta) while also exploring the environment.

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 [12], [13]. Other algorithms, however, add more rules or complex control on top of Boids to improve it [14]–[16].

In this work, our goal is to design swarm robotic algorithms to safely explore an environment while also 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 avoid collisions with other agents and obstacles in the environment.
- **Tunable:** The algorithm should be tunable to trade-off exploration and exploitation.

Our main contribution is a novel algorithm that extends the traditional Boids algorithm to meet these objectives. 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 in the same way as 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 towards the target location versus how much it explores the environment. To further improve safety, we use control barrier functions [17] to minimally modify the algorithm to avoid collisions at an additional computational cost. While this does not guarantee safety, we show through numerical simulations that safety is vastly improved, even in large swarms.

Figure 1 illustrates our results. Here, a group of 100 agents explores an environment consisting of a square boundary and two walls. One agent is given the location of a goal, and the agents use local interactions to both explore the environment and move towards the goal. To visualize this behavior, we use a heatmap of the agent locations aggregated over time for multiple simulations, where darker regions correspond to well-explored areas. In this scenario, the agents effectively explore the entire environment; 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 II. We then introduce the two main components of our algorithm, ghost boids and control barrier functions, in Sections III and IV. We describe our results in Section V, and we provide concluding remarks in Section VI.

II. 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 Figure 1. Once an agent collides with another agent or an obstacle, it is considered "dead" and becomes stationary.

A. Simulation parameters

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

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

B. Performance metrics

Recall that the goal is for the group of 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.

a) 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.

b) Performance: We have two metrics to characterize how well a group of agents explores an environment while also 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 the target is within its neighbor radius at any point during the simulation, even if the agent later moves away from the target and/or has a collision. 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 that were searched many times over the simulations (possibly by multiple agents), while lighter regions indicate low search coverage. An example heatmap is shown in Figure 1.

C. 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 Figure 2.

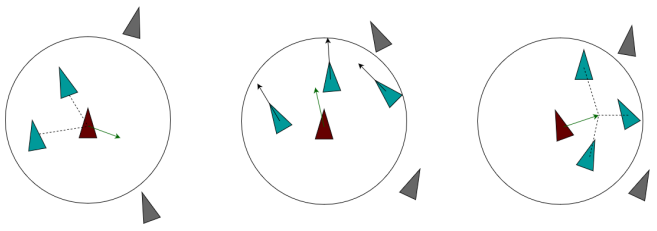


Fig. 2: Visualization of Boids rules. Left to right: separation, alignment, and cohesion.

To motivate the need for obstacle-avoidance and goal-seeking behavior, consider running the Boids algorithm in a

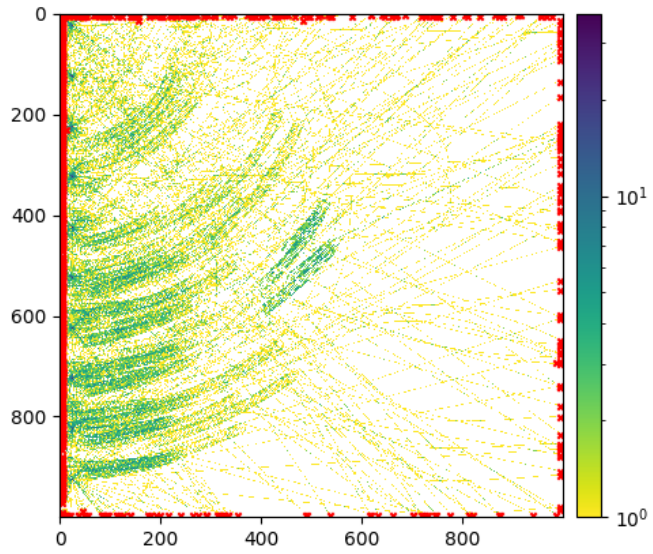


Fig. 3: The traditional boids algorithm has no obstacle-avoidance or goal-seeking behavior.

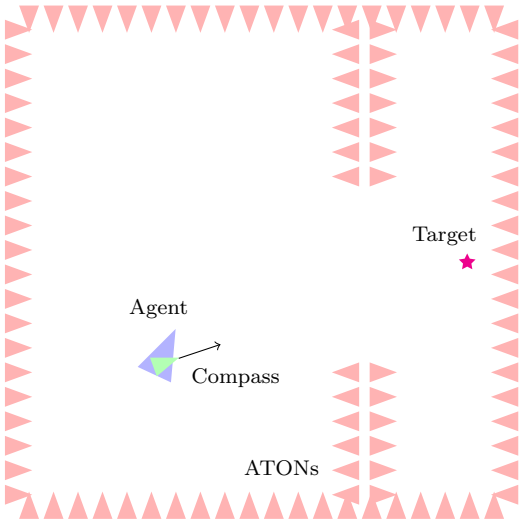


Fig. 4: Illustration of ghost boids. ATONs (red triangles) point away from danger for collision avoidance. A compass (green) is located on an informed agent (blue) and points toward the target (magenta) for goal-seeking behavior.

bounded environment with no obstacles. The corresponding heatmap is shown in Figure 3, where agents wander through the environment until colliding with the boundary (collisions shown in red). We describe our extensions to this algorithm that address these issues in the next two sections.

III. 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 in character to normal agents, but they do not actively move on their own. Ghost boids affect the rules for separation, alignment, and cohesion for neighboring agents. There are two types of ghost boids: aids to navigation (ATONs) [18] and compasses, which we now discuss.

A. Aids to Navigation (ATONs)

We use ATONs to provide obstacle avoidance. ATONs are placed around the perimeter of obstacles and the environment as shown in Figure 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 both repels the agent from its location while also aligning the heading of the agent away from danger.

B. Compasses

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

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 Figure 4 (green). To avoid the influence of the compass being attenuated by a large number of neighbors, we scale the influence (or weight) of the compass proportional to the number of neighboring agents and ATONs (but not other compasses).

IV. Control Barrier Functions

While both 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 Figure 5, with many collisions occurring both at obstacles and in the interior of the environment between agents.

To further improve safety, we use a control barrier function (CBF) [17] to modify the algorithm in a computationally efficient way that promotes safety. 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. A state may be safe, for instance, if no agent is colliding with an obstacle or any other agent. Suppose we can describe the safe set as the super-level set of some barrier function h applied to all pairs of boids,

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

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

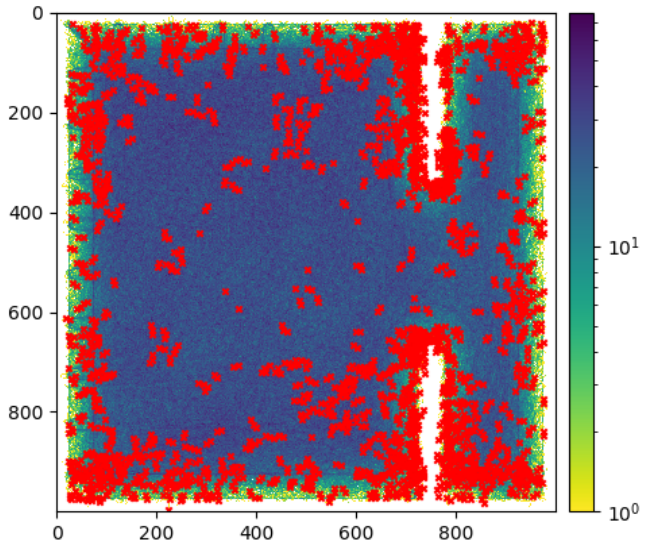


Fig. 5: Heatmap of the boids algorithm with ghost boids. Even with the separation rule and ATONs, many collisions occur (red).

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 such that the barrier function does not become too small so that 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 time derivative of the barrier function along the dynamics of the system. Alternatively, in discrete time the safety constraint is

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

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:

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

If the nominal velocity v_i^{nom} satisfies the safety constraint, then it is trivially the optimal solution. If not, the optimizer finds the velocity that is 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 (QCQP), 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 instead propose a modified problem that is easier to solve but that does not guarantee safety. Define the positive constant $\alpha = \Delta t^{-1}(1 - c)$. For Δt sufficiently small, we can neglect the quadratic term in the above inequality 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 quadratic program, which is convex and can be solved efficiently using standard numerical solvers [19]–[22]. 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. 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. We show in our simulations, however, that 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 QCQP.

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.

[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.

V. Experimental results

We now illustrate the properties of our algorithm using simulations of the search and rescue scenario described in Section II under various parameter settings.

A. Safety vs efficiency

We first study the trade-off between safety and computational efficiency. Figure 6 shows the heatmap produced by 10 agents, none of which are informed, in an open environment.

The most efficient is the traditional Boids algorithm, shown in Figure 6a, 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 Figure 6b. Now, most agents continue to wander through the environment over the course of the simulation, resulting in a darker heatmap. The repulsive affect of the separation rule, however, 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 Sec. IV 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, so 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 I for a varying number of agents. The traditional Boids algorithm is most computationally efficient but the least safe, while the CBF algorithm is most safe but least efficient. All algorithms become less safe and have longer runtimes as the number of agents increases.

TABLE I: 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	(11352)	99.9	(25528)

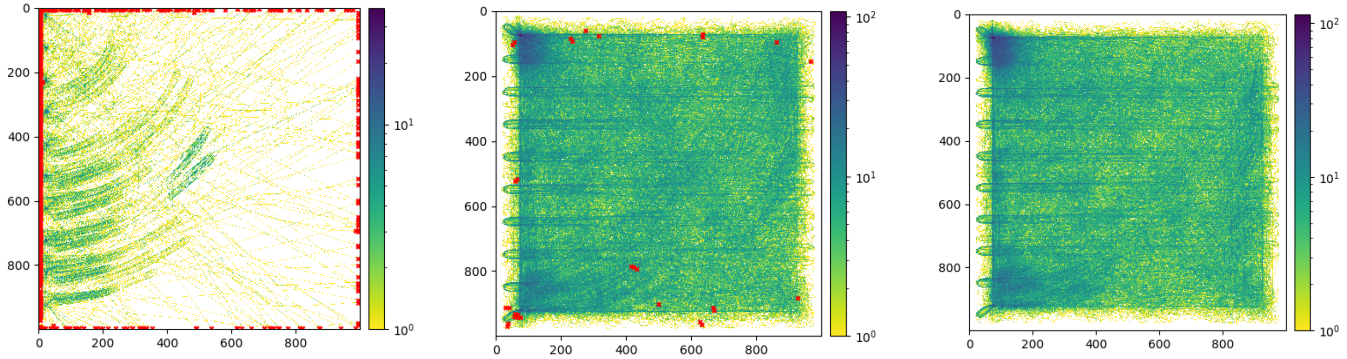
B. Exploration vs exploitation

We now study the trade-off between exploring the environment and exploiting information about the possible target location that is available to informed agents. There are two parameters that 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 Figure 7 for varying amounts of compass influence. Without the compass (Fig. 7a), the agents uniformly cover the environment with few collisions due to the CBF mechanism. With a small amount of compass influence (Fig. 7b), the agents still explore the environment, but to a lesser extent as they spend more time near the possible target location due to the compass pointing in that direction. With 100% compass influence (Fig. 7c), 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 results 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 obstacles in the environment. Consider the same setup as before, but in an environment with a bowl-shaped obstacle as illustrated in Figure 8. Here, the

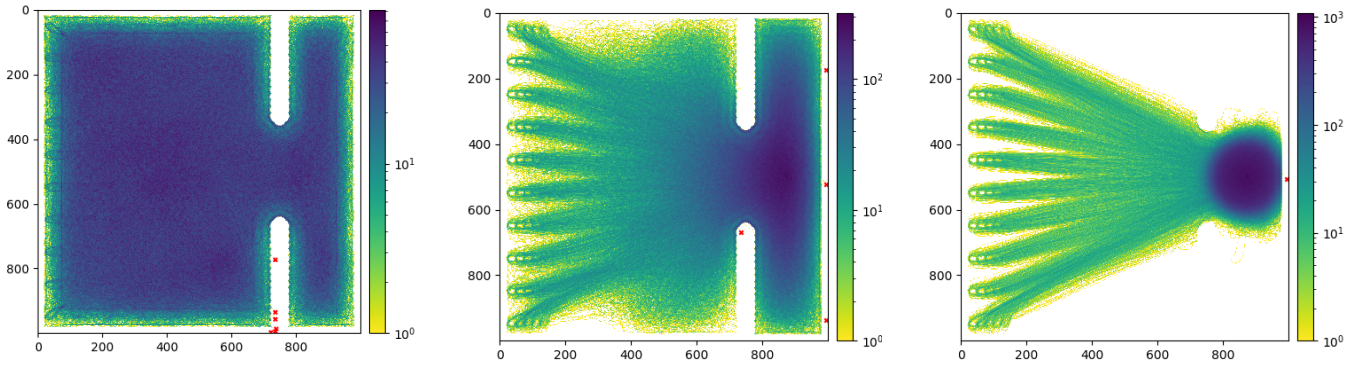


(a) The traditional boids algorithm has many collisions at the boundary.

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

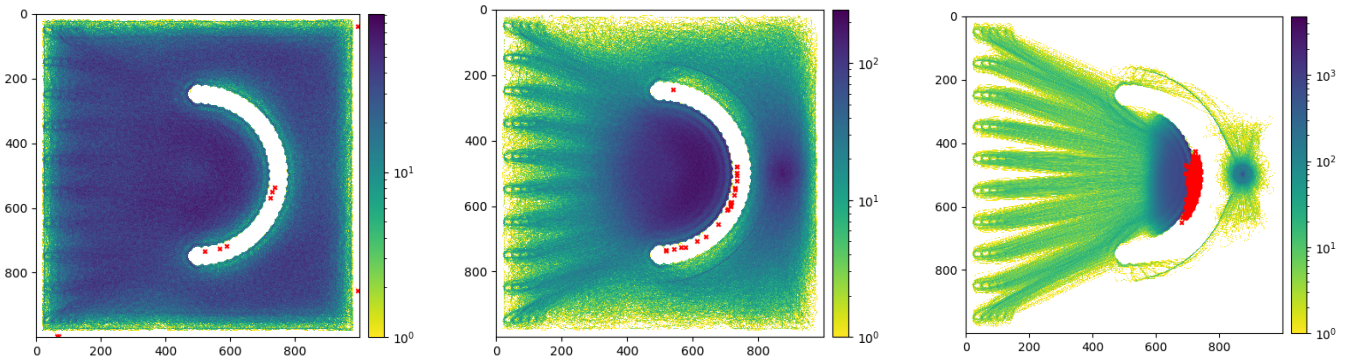
(c) The CBF algorithm has no collisions, but agents solve a QP at each iteration.

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. 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 trapped in the bowl.

target is located on the opposite side of the bowl as the starting locations 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 a large number of collisions due to the high density of agents inside the bowl. We conclude that complex environments require small compass influence so that agents explore enough of the environment to locate the target.

VI. 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, and then 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 the use of ATONs and CBFs, and that the number of informed agents and compass influence can be used to trade-off exploration of the environment with exploitation of possible target location information. In complex environments (such as that with the bowl-shaped obstacle), however, too much compass influence can cause livelock scenarios in which informed agents are unable to navigate around the obstacle to reach the target.

Our algorithm could be extended in various ways. We made several simplifying assumptions in the derivation of the safety constraint for the CBF, so our algorithm does not guarantee safety. An interesting problem is how to construct a safety constraint for the discrete-time dynamics that is always feasible so that safety can be guaranteed. 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 the case where all informed agents have access to the same single 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 of these possible target locations with the rest of the environment. Beyond a finite number of single points, we may in general 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 an amount of time in each region proportional to the probability that the target is in that region. And 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.

REFERENCES

[1] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," in *International Conference on Emerging Security Technologies*, pp. 142–147, 2010.

[2] J. L. Baxter, E. Burke, J. M. Garibaldi, and M. Norman, "Multi-robot search and rescue: A potential field based approach," *Autonomous Robots and Agents*, pp. 9–16, 2007.

[3] J. León, G. A. Cardona, A. Botello, and 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*, (Porto, Portugal), pp. 1061–1070, Springer, Dec 2017.

[4] A. Matos, A. Martins, A. Dias, B. Ferreira, J. M. Almeida, H. Ferreira, G. Amaral, A. Figueiredo, R. Almeida, and F. Silva, "Multiple robot operations for maritime search and rescue in euRathlon 2015 competition," in *OCEANS*, (Shanghai, China), pp. 1–7, 2016.

[5] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen, *Search and Rescue Robotics*. Berlin, Heidelberg: Springer, 2008.

[6] K. J. Rafferty and E. W. McGookin, "An autonomous air-sea rescue system using particle swarm optimization," in *International Conference on Connected Vehicles and Expo*, pp. 459–464, 2013.

[7] Z. Chen, H. Liu, Y. Tian, R. Wang, P. Xiong, and G. Wu, "A particle swarm optimization algorithm based on time-space weight for helicopter maritime search and rescue decision-making," *IEEE Access*, vol. 8, pp. 81526–81541, 2020.

[8] P. Xiong, L. Hu, T. Yongliang, C. Zikun, W. Bin, and Y. Hao, "Helicopter maritime search area planning based on a minimum bounding rectangle and k-means clustering," *Chinese Journal of Aeronautics*, vol. 34, no. 2, pp. 554–562, 2021.

[9] B. Ai, M. Jia, H. Xu, J. Xu, Z. Wen, B. Li, and D. Zhang, "Coverage path planning for maritime search and rescue using reinforcement learning," *Ocean Engineering*, vol. 241, p. 110098, 2021.

[10] W. Yue, Y. Xi, and X. Guan, "A new searching approach using improved multi-ant colony scheme for multi-UAVs in unknown environments," *IEEE Access*, vol. 7, pp. 161094–161102, 2019.

[11] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *SIGGRAPH Computer Graphics*, vol. 21, p. 25–34, Aug 1987.

[12] P. M. Mavhemwa and I. Nyangani, "Uniform spatial subdivision to improve boids algorithm in a gaming environment," *International Journal for Advance Research and Development*, vol. 3, no. 10, pp. 49–57, 2018.

[13] J. Hagelbäck, "Hybrid pathfinding in StarCraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 8, no. 4, pp. 319–324, 2015.

[14] J. B. Clark and D. R. Jacques, "Flight test results for UAVs using boid guidance algorithms," *Procedia Computer Science*, vol. 8, pp. 232–238, 2012.

[15] J. Wang, H. Zhao, Y. Bi, S. Shao, Q. Liu, X. Chen, R. Zeng, Y. Wang, and L. Ha, "An improved fast flocking algorithm with obstacle avoidance for multiagent dynamic systems," *Journal of Applied Mathematics*, vol. 2014, 2014.

[16] B. N. Sharma, J. Vanualailai, and J. Raj, "Obstacle and collision avoidance control laws of a swarm of boids," *International Journal of Mathematical, Computational Science and Engineering*, vol. 8, pp. 253–258, 2014.

[17] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *18th European Control Conference*, pp. 3420–3431, 2019.

[18] United States Coast Guard, 13th District, Office of Boating Safety, "U.S. Aids To Navigation System," 2003.

[19] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[20] G. Banjac, P. Goulart, B. Stellato, and S. Boyd, "Infeasibility detection in the alternating direction method of multipliers for convex optimization," *Journal of Optimization Theory and Applications*, vol. 183, no. 2, pp. 490–519, 2019.

[21] G. Banjac, B. Stellato, N. Moehle, P. Goulart, A. Bemporad, and S. Boyd, "Embedded code generation using the OSQP solver," in *IEEE Conference on Decision and Control*, 2017.

[22] B. Stellato, V. V. Naik, A. Bemporad, P. Goulart, and S. Boyd, "Embedded mixed-integer quadratic optimization using the OSQP solver," in *European Control Conference*, 2018.