Target-oriented predator and prey swarm control in obstacle-filled environments

Catalin-Petru Lupau
Responsible Professor: Dr. Ranga Rao Venkatesha
Supervisor(s): Ashutosh Simha, Suryansh Sharma
EEMCS, Delft University of Technology, The Netherlands

1 Abstract

In this paper, we explore the creation of control algorithms for swarms of robots playing the role of either predator or prey in an environment filled with static obstacles. The paper develops on a famous flock simulation model proposed by Craig Reynolds called *boids*. The paper analyzes a zero-sum game situation, in which one swarm of robots, the prey, is trying to reach a certain pre-determined target, while another swarm of robots, the predator, is trying to prevent it from reaching its objective. Swarm control algorithms for both the predator and the prey scenarios are analyzed in an arms race manner. The robots are modelled as point-mass holonomic entities, that can move in arbitrary directions. The proposed algorithms are tested on characteristics such as success rate and time in a simulated environment. As a result, a set of algorithms for both the predator and prey are proposed and their strength and weaknesses are discussed.

Keywords: predator-prey, obstacle-avoidance, swarm control, boids, UAVs, robot swarms, war robots, military robots, self-driving cars, collision avoidance, simulation, control algorithm

2 Introduction

2.1 Background

A swarm can be defined as a collection of entities interacting with each other through simple pre-defined rules from which a highly complex behaviour emerges. Swarm behaviour is encountered in nature at various species of animals, ranging from fishes to mammals and birds, and it usually serves an optimisation purpose. The fact that swarm behaviour evolved independently in so many species, indicates that being able to recreate it might be beneficial for solving various kinds of real-life problems [2].

One particular context in which the exploration of swarm behaviour could lead to interesting and useful results is a predator-prey game. A predator-prey game, or a 'predation' represents a situation in which a group of individuals from one species is trying to kill the individuals from another species [1]. This creates a conflict between 2 groups of agents, the predators and the prey. Since the 2 groups of agents have fully conflicting interests, the entire situation can be seen as a zero-sum game. This means, that any advantage gained by one party represents an equal disadvantage to the other party.

The particular type of predator-prey game that this paper explores is target-oriented. This means that the purpose of the prey is not only to survive the predators, but also to move to a certain target in the environment. The environment that the prey needs to navigate is not empty, however, but rather it contains obstacles of various sizes at random positions. Thus, the prey swarm needs to successfully navigate an obstacle-filled environment, while at the same time avoiding predators. In contrast, the predator swarm has the opposite goal of preventing the prey from reaching the desired target.

2.2 Importance

As highlighted before, insight into swarm control algorithms in the context of a predator-prey situation could lead to useful applications.

One such application is a war-like scenario involving robots. Concretely, let us say that we have a swarm of UAVs with the mission of reaching a spatial target to deliver supplies. There are certain spots along the way that the UAVs should avoid, since in these spots, the UAVs are vulnarable to enemy fire from below. On top of that, the enemy deploys his/her own swarm of Kamikaze UAVs, with the purpose of taking down the original UAVs before reaching their target [15]. This situation easily fits the context of target-oriented predator-prey swarm control, since the friendly UAVs can be modelled as a prey swarm, the kamikaze UAVs as a predator swarm and the dangerous spots as environment obstacles.

Applications are not strictly limited to war-like scenarios. Research into this area could prove useful for robotics in a more general sense. Taken separately, the prey swarm control algorithm could be used in the context of any fleet of robots trying to navigate an obstacle-filled environment in order to reach a certain destination safely. As a matter of fact, navigation throughout an area that contains obstacles is a common problem in robotics [5]. Since we want to avoid collisions in-between the robots of the fleet, it makes sense to model the fleet as a swarm. Unexpected moving obstacles might be part of the navigated environment. In such a case the anti-predator evasive moves of the prey swarm control algorithm are of great use.

To illustrate the previous point with a concrete example, let us look at self-driving cars. Up to a certain point, several cars driving on a road share a common destination, for example, the exit of the highway. It is crucial for the self-driving cars to avoid colliding into each other. There might be certain static obstacles on the road or high-way, such as a section that is under construction, or a stationary car. Moving obstacles are also possible, such as humans or animals traversing the road. Since each self-driving car is its own distinct entity, centralized decision making is not possible. It is not hard to see that swarm control algorithms, especially the kind that can avoid predators (humans or animals walking towards the self-driving car), can prove very useful in this context [7].

On another note, since both swarm behaviour and predator-prey scenarios are inspired by nature, the algorithms presented in the paper might be used to create graphical simulations of a group of predators trying to hunt down prey, such as a pack of wolves hunting dear, or banks of fish hunting other fish [10]. Realistic graphics simulation of such phenomena could then be used in movies or video games.

2.3 Problem Formalization

In order to have a good understanding of the problem that this paper is exploring, it is of great use to lay down the main concepts that are being dealt with, which is what the following subsections do. The appendix provides a graphical representation of the problem environment.

Safe area

The safe is the area where the prey agents are initially spawned, on the right side of the map. The spawn positions of the prey agents are random. No obstacles spawn in the safe area.

Danger area

The danger area is the area the prey swarm needs to traverse in order to reach the target area. It contains obstacles that need to be avoided, as well as predators that can destroy prey agents.

Target area

The target area is the area that all prey agents need to reach. It is located on the left side of the map. Predator agents that hit the target area get destroyed. This mechanic is to prevent predator agents from camping around the target area (our main interest is intercepting the prey while they are on the move).

Obstacles

The obstacles are regions inside the danger area that the swarm agents need to avoid on their way to the target area. They spawn at random positions, and they have random sizes. If either a prey agent or predator agent hits an obstacle, that agent gets destroyed.

Prey agents

The prey agents are the agents that need to reach the target area. They spawn at random positions in the safe area (right side of the map) and on their way to the target area (left side of the map), they need to avoid obstacles and predators. This paper explores several swarm control algorithms for the prey agents.

Predator agents

The predator agents form a swarm that has the goal of preventing the prey swarm from reaching the target. The predator agents act by crashing into the prey agents, destroying them. On impact with a prey agent, the predator gets destroyed as well. Predators also get destroyed if they hit the target area. This paper explores several swarm control algorithms for the predator agents.

Perception

Each prey agent is aware of the position of the target area and obstacles, as well as of the positions and velocities of all other prey agents in the swarm. In contrast to this, prey agents can only detect predator agents that are within their perception distance. Predator agents have an advantage over the prey. On top of everything else in the environment, predators are always aware of the positions and velocity of the prey agents.

Goals and constraints

The goal of the prey swarm control algorithm is to maximize the number of prey agents that reach the target area, while at the same time minimizing the time it takes for them to get there. The prey agents need to try to avoid all obstacles, since crashing into an obstacle disables the agent. The predator agents, on the other hand, need to prevent the prey agents from reaching their goal, by crushing into them, so a lower performance of the prey is seen as a higher performance of the predator. The top speeds of both the prey agents and predator agents are limited.

2.4 Research Questions

All of the above lead us to the questions that this paper aims to answer:

- Can the number of prey agents that reach the target be increased and their travel time be decreased through the creation or use of smart prey swarm control algorithms? If so, how would such a algorithms work? How would they compare with each other?
- Given some prey swarm control algorithm, can the number of prey agents that reach the target be decreased or their travel time be increased through the creation or use of smart predator swarm control algorithms? If so, how would such a algorithms work? How would they compare with each other?

3 Related Work

3.1 Spring and dampers swarm control

Wiech, Jakub, Victor A. Eremeyev, and Ivan Giorgio discuss a method of swarm control based on a pair of virtual spring and dampers used to create a structure between the agents [16]. The springs and dampers make sure that the swarm agents do not collide, while at the same time preserving the cohesion of the swarm.

The control algorithm tackles obstacle avoidance is a similar fashion, by modelling the interaction between a swarm agent and a nearby obstacle using the same type of parallel spring and damper system.

From a mathematical perspective, a spring applies a force directly proportional to its elongation. The direction of the applied force is alongside the main axis of the spring.

On the other hand, the force generated by the damper is directly proportional to the 'elongation speed', or the derivative of the elongation.

By adding the 2 forces, we obtain the force generated by one instance of the parallel spring-damper system, as seen in equation 1.

$$F = F_{spring} + F_{damper} = k(l - l_0) + c\frac{\mathrm{d}(l - l_0)}{\mathrm{d}t} \quad (1)$$

Each agent is connected to its swarm members by a spring-damper system acting with the force given by equation 1. By computing the resulting force and then dividing by the mass of the agent, we can obtain the formula for the acceleration that the agent needs to move with.

$$\vec{a}_i = \frac{1}{m_i} \sum_{j \in N_i} \vec{F}_{ij} \quad (2)$$

As pointed out by the authors, this control algorithm causes the agents to oscillate. Such oscillations can be undesirable when more precise control of the behaviour of each agent is wanted, which is why this model was not chosen as the theoretical background of this paper. While the oscillations can be tackled by good hyper-parameter optimization, they cannot be fully prevented due to the nature of the dynamical model.

3.2 Boids

Craig Reynolds attempted to find an easy and computationally efficient way to simulate natural swarms, such as flocks of birds, herds of land animals or schools of fish [11]. The result of his research was a novel computational model, called 'boid', representing a simplistic form of 'artificial life' that can realistically simulate flocking behaviour using a set of very simple dynamical laws.

As C. Hartman and B. Benes point out [6], the 3 main laws that govern the movement of a boid are:

- **Cohesion** - defined as the boid's tendency to stick to the center of its flock.
- **Separation** - defined as the boid's tendency to keep a certain distance from its swarm neighbors in order to avoid collision.
- **Alignment** - defined as the boid's tendency to synchronize its position and direction with the other members of the swarm/flock.

The implementation of the 3 laws above is enough to create a functional swarm of boids, however, such a swarm is not very useful without a goal. Luckily, the boids model is flexible and can be extended in various ways, such as adding a steering force or obstacle avoidance mechanisms in order to get the boids to achieve a certain goal or task.

Due to its simplicity and extensibility the boids model was chosen as the basis for the algorithms developed in this paper. The mathematical description of the model is presented in the theoretical background section.

3.3 Fear model for boids

Carlos Delgado-Mata, Jesus Ibanez Martinez, Simon Bee, Rocio Ruiz-Rodarte and Ruth Aylett propose an interesting method of emulating animal fear in a flock of boids [4]. While their research is not the topic of this paper, their idea of having the prey communicate the presence of a predator acted as a source of inspiration for the explosive predator avoidance strategy.

4 Methodology

4.1 Algorithms development

The methodology used for algorithm development is adapted to and reflects the nature of the problem that this paper deals with. The algorithms that need to be explored fall into 2 categories. The first category is represented by the prey swarm control algorithms, which need to maximize the number of prey agents that reach the target while minimizing the time it takes for them to do so. In contrast to that, the second category consists of the predatory control algorithms having a conflicting goal. While being different, the 2 categories of algorithms share a common component, which is obstacle avoidance.

The intrinsic structure of the problem leads down to a natural breakdown of the methodology. The first step consists of developing the obstacle avoidance algorithms, which are shared sub-components of both prey and predator control. After this class of algorithms is fully explored and benchmarked, the next step is to move on to the actual prey-predator swarm control. Since these 2 categories can be seen as parties in a zero-sum game, an 'arms race' methodology is followed.

The 'arms race' approach consists of an iterative process of improvement, alternating from prey to predator control, like in an actual 'arms race'. This means that first, prey control algorithms are developed. Afterwards, predator control algorithms capable of countering the prey control algorithms are created. Then, we move on to prey control again. This process is repeated until state-of-the-art solutions are obtained.

4.2 Simulation and benchmarking

The algorithms developed according to the process described above need some way to be evaluated and benchmarked against each other. Since performing benchmarks on real hardware would be difficult, the natural solution is to use a simulated environment. The simulated environment developed for this research consists of two tools, with complementary functions:

- **Headless Simulator** - takes configuration files defining the simulations to be performed as input and outputs simulation files, as well as their results as plots.
- **Web-based simulation visualizer** - takes simulation files as input and creates visualizations of them.

The simulation environment was implemented in python, using 'flask' as a backend framework. The simulation environment used in this research is open source. The appendix contains the link to the repository.

5 Theoretical Background

As presented in the related work section, the computational model based on which all swarm control algorithms in this paper are implemented is the 'boids model'. As C. Hartman and B. Benes present in their paper [6], the following mathematical characterization can be formulated for the boids (all the formulas are taken from the paper):

5.1 Boid

Each boid in this paper is characterized by a position $\vec{p_i}$ and a velocity $\vec{v_i}$, but not an orientation, as in the original paper. This leads to formally defining the boid as a pair of the 2 quantities, as in equation 3

$$b_i = (\vec{p_i}, \vec{v_i}) \tag{3}$$

A swarm of boids is, then, simply a set of boids that are interacting with each other.

The neighbors of a certain boid b_i represent the boids b_j of the swarm that are within a certain distance d from b_i.

$$N_i = \{b_j \mid \|\vec{p_i} - \vec{p_j}\| < d\} \tag{4}$$

5.2 Separation

Separation represents the force that prevents 2 or more boids of the same swarm to collide with each other, by pushing them aside when they get too close. Separation is mathematically defined in equation 5.

$$\vec{s_i} = -\sum_{b_j \in N_i} (\vec{p_i} - \vec{p_j}) \tag{5}$$

5.3 Cohesion

Complementary to the separation force, the cohesion force has the role of keeping the swarm together. This means that this force will determine each agent to move towards the mean position of its neighbors. This is expressed mathematically in equation 6 (the mean position) and equation 7 (the actual cohesion force)

$$\vec{c_i} = \frac{1}{|N_i|} \sum_{b_j \in N_i} \vec{p_j} \tag{6}$$

The cohesion force, thus, becomes:

$$\vec{k_i} = (\vec{c_i} - \vec{p_i}) \tag{7}$$

5.4 Alignment

The role of this force is to make sure that the velocities of the swarm somewhat match each other, so that the swarm moves roughly in the same direction. This force is inspired by the natural tendency of flock animals to try to match their velocity to the velocity of their peers. Mathematically, we can express the alignment force as follows:

$$\vec{m_i} = \frac{1}{|N_i|} \sum_{b_j \in N_i} \vec{v_j} \tag{8}$$

5.5 Combining the steering forces

All the steering forces applicable on agent i can be linearly combined into one steering force, as follows:

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m_i} \tag{9}$$

The optimal values for the real parameters S, K, M depends on the problem, and can be determined using optimization techniques, such as genetic algorithms [3].

Each time instance the steering force increments the value of the boid's velocity, as shown in equation 10.

$$\vec{v_i} := \vec{v_i} + \vec{f_i} \tag{10}$$

Equation 11 shows how the position of the boid gets incremented using the velocity in time instances of size Δt.

$$\vec{p_i} := \vec{p_i} + \vec{v_i}\Delta t \tag{11}$$

6 Algorithms Description

6.1 Static Obstacle Avoidance

One of the ways in which the boids model needs to be extended in order to solve the proposed problem is the addition of obstacle avoidance capabilities. As presented in the problem formalization section, the area between the spawn point of the prey and the target is filled with randomly generated static obstacles. A swarm agent, either a prey or predator, hitting any of these obstacles gets destroyed. Thus, the success rate of any of the 2 categories of swarm control algorithm depends on the chosen obstacle avoidance strategy. The following sections will explain in detail the various obstacle avoidance strategies proposed by this paper.

Outwards force field

Following the direction indicated by the original boids paper, one of the obstacle avoidance models that this paper proposes is a force field radiating outwards from each obstacle [11]. The proposed outwards force field takes the gravitational force field as a source of inspiration, but rather than radiating inwards, as is the case for gravity, this force field radiates outwards, rejecting any agents that come too close [8].

Since we are modelling the outwards force field as an inverse gravitational field, we want the strength of the field to be inversely proportional to the square distance of the agent with respect to the center of the field. We can thus represent the force field as a vector field with the following formula, expressed in a coordinate system tied to the static obstacle.

$$\vec{g}(\vec{r}) = \frac{w_g}{\|\vec{r}\|^2}\vec{r} \tag{12}$$

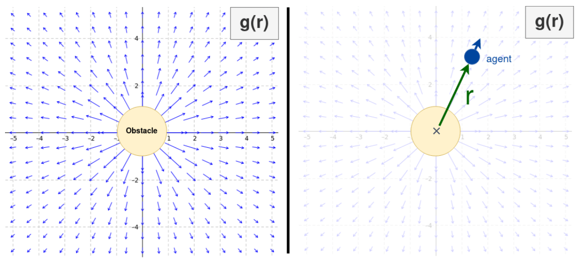


Figure 1: (Left) Outwards force field around obstacle. (Right) Agent located in an outwards force field.

The formula defines a force field that acts on agents regardless of their distance to the static obstacles. Even though the strength of the force field decreases with the distance, we would ideally want to confine the force field within a certain area around the static obstacle. This helps avoid unnecessary movements caused by obstacles far away from the agent and, hence, of no risk of collision with it. As a result of this, the extension to the previous formula in equation 13 is proposed.

$$\vec{g}(\vec{r}) = \begin{cases} \frac{w_g}{\|\vec{r}\|^2}\vec{r}, & \text{if } \|\vec{r}\| \leq d \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

The last aspect that we need to see is how to introduce the force field into our pre-existing model of the boid. This is achieved by adding the sum of all force fields as a weighted term to the boid steering force defined initially in equation 17.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m_i} + G\sum_{j \in O} \vec{g}(\vec{p_i} - \vec{o_j}) \tag{14}$$

where: G = real coefficient
$\vec{p_i}$ = the position of boid i
$\vec{o_j}$ = the position of obstacle j

Spiral force field

The outwards force field presented in the previous section has one big disadvantage. When the agent approaches it frontally, rather than altering the agent's direction, it only slows the agent down, which is not desirable for achieving obstacle avoidance.

The attempt this paper proposes with regard to tackling the issue is a spiral force field. This means that rather than having a force field radiating outwards, we alter its direction to make it radiate in a spiral shape, essentially creating a reverse 'whirlpool'. The formula in equation 15 corresponds to generating such a force field.

$$\vec{h}(\vec{r}) = (r_x - r_y)\vec{i} + (r_x + r_y)\vec{j} + \frac{w_h}{\|\vec{r}\|^2}\vec{r} \tag{15}$$

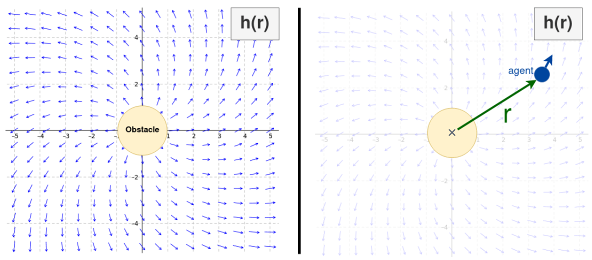


Figure 2: (Left) Spiral force field around obstacle. (Right) Agent located in a spiral force field.

As in the case of the outward force field, we would like to be able to clamp the force field once the agent is outside a certain area around the obstacle. In a similar fashion, this can be achieved by extending the formula of the force field in the following way.

$$\vec{h}(\vec{r}) = \begin{cases} (r_x - r_y)\vec{i} + (r_x + r_y)\vec{j} + \frac{w_h}{\|\vec{r}\|^2}\vec{r} \leq d \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

Integrating the spiral force field into the boids computational model is done in a similar fashion as for the outwards force field.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m_i} + H\sum_{j\in O}\vec{h}(\vec{p_i} - \vec{o_j}) \tag{17}$$

Directional spiral force field

One issue with the spiral force fields presented in the previous subsection is that, if two obstacles are relatively close to each other, the rotating component in the region between 2 obstacles might get cancelled.

The directional spiral force field tries to solve this, by adjusting the direction of the spirals, based on the boid's steering direction. Rather than cancelling each other, the 2 force field should act together to provide a velocity boost to the boid. Equation 20 shows how this can be achieved by alternating the direction of the spiral as either clockwise (equation 19) or counter-clockwise (equation 18), based on the velocity of the boid and the relative position of the obstacle to the boid.

$$\vec{h}_a(\vec{r}) = (r_x - r_y)\vec{i} + (r_x + r_y)\vec{j} + \frac{w_h}{\|\vec{r}\|^2}\vec{r} \tag{18}$$

$$\vec{h}_c(\vec{r}) = (r_x + r_y)\vec{i} + (r_y - r_x)\vec{j} + \frac{w_h}{\|\vec{r}\|^2}\vec{r} \tag{19}$$

$$\vec{h}_{dir}(\vec{r},\vec{s}) = \begin{cases} \vec{h}_a(\vec{r}), & \|\vec{r}\| \leq d \wedge \vec{s}\cdot\vec{r} < 0 \\ \vec{h}_c(\vec{r}), & \|\vec{r}\| \leq d \wedge \vec{s}\cdot\vec{r} \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

The directed spiral force field is integrated with the boid's model in the same way as the spiral force field, as shown by equation 21.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m_i} + H\sum_{j\in O}\vec{h}_{dir}(\vec{p_i} - \vec{o_j}, \vec{t} - \vec{p_i}) \tag{21}$$

Steer away strategy

The other direction in terms of collision avoidance that the original paper on boids recommends is trying to avoid obstacles by steering away from imminent collisions [11]. Pursuing the recommendation, this paper proposes an algorithm that always assumes a constant trajectory for the boid and, if an obstacle is found on the assumed constant trajectory, the boid's trajectory is slightly altered.

Figure 3 shows the boundaries of the agent's trajectory within which the obstacle will be hit. The first step that needs to be taken, in order to determine whether or not the agent will hit the obstacle is finding those boundaries. This is equivalent to finding the angle between the collision axis (axis uniting the center of the agent with the center of the obstacle) and one of the boundaries. This angle, named α is drawn in Figure 3.

By applying simple geometry based on the structure presented in figure 3, we obtain the following formula for the tangent of the angle α:

$$\tan(\alpha) = \frac{S_1}{l_1} = \frac{S_2}{l_2} \tag{22}$$

From equation 22 we can, then, derive the following:

$$S_1 + S_2 = \tan(\alpha)(l_1 + l_2) \iff \tag{23}$$

$$\tan(\alpha) = \frac{S_1 + S_2}{l_1 + l_2} \iff \alpha = \tan^{-1}(\frac{S_1 + S_2}{l_1 + l_2}) \tag{24}$$

Now that the boundary trajectories are known, we need to determine whether or not the agent's trajectory is within those boundaries. If it is the case, then we know that the obstacle is on a collision course and, thus, the trajectory of the agent

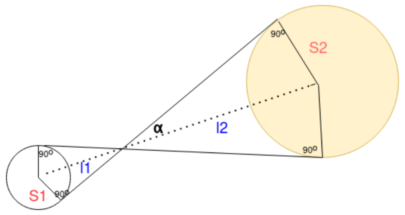


Figure 3: Geometry of the maximum collision angle. Yellow - Obstacle, White - Agent

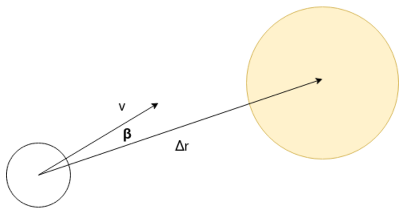


Figure 4: Geometry of the trajectory angle. Yellow - Obstacle, White - Agent

needs deviation. So let us see how the trajectory angle β, that can be seen in figure 4 can be determined.

$$\tan(\beta) = \frac{\sin(\beta)}{\cos(\beta)} = \frac{\frac{|\vec{v}\times\Delta\vec{r}|}{|\vec{v}||\Delta\vec{r}|}}{\frac{\vec{v}\cdot\Delta\vec{r}}{|\vec{v}||\Delta\vec{r}|}} = \frac{|\vec{v}\times\Delta\vec{r}|}{\vec{v}\cdot\Delta\vec{r}} \tag{25}$$

Based on the derived equations, we can formulate the mathematical condition for when the agent is on a collision course with an obstacle. As it can be geometrically seen in the figures 3, 4, in order for the agent to be on a collision course with the obstacle the trajectory angle needs to be within the boundary angle. This translates mathematically to the following.

$$-\alpha \leq \beta \leq \alpha \iff \tag{26}$$

$$-\tan(\alpha) \leq \tan(\beta) \leq \tan(\alpha) \iff \tag{27}$$

$$-\frac{S_1+S_2}{l_1+l_2} \leq \frac{|\vec{v}\times\Delta\vec{r}|}{\vec{v}\cdot\Delta\vec{r}} \leq \frac{S_1+S_2}{l_1+l_2} \iff \tag{28}$$

$$\frac{|\vec{v}\times\Delta\vec{r}|}{|\vec{v}\cdot\Delta\vec{r}|} \leq \frac{S_1+S_2}{l_1+l_2} \tag{29}$$

So we know that the agent is on a collision course whenever $\frac{|\vec{v}\times\Delta\vec{r}|}{|\vec{v}\cdot\Delta\vec{r}|} \leq \frac{S_1+S_2}{l_1+l_2}$ holds. The next step is to develop an update rule that enables the agent to steer away from the obstacle in order to avoid collisions. Let γ represent the value of the small angle adjustment that we want to make to the current course. Since we always want to steer towards the direction that will lead to the fastest avoidance, our steer will be positive whenever the current course is 'above' the collision axis and negative otherwise. Using a 2D rotational matrix $R(x)$, we can mathematically encode the update rule as follows.

$$\vec{v} := \begin{cases} R(\gamma)\vec{v}, & \text{if } \vec{v}\cdot\Delta\vec{r} \geq 0 \\ R(-\gamma)\vec{v}, & \text{otherwise} \end{cases} \tag{30}$$

Kitchen sink approach

Since the force field methods and the steer away method are not mutually exclusive, the kitchen sink approach proposes implementing a combination. More precisely, in the kitchen sink approach, the boid makes use of both the directional spiral force field and the steer away strategy in order to avoid collisions.

6.2 Prey Control Algorithms

Another direction in which the boids swarm model presented so far can be extended is as a prey swarm. This means that the steering force needs to be modified in such a way that the boids move towards the target area, as shown in equation 31.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m_i} + \text{col. avoid. term} + T(\vec{t} - \vec{p_i}) \tag{31}$$

While simply moving towards the target area might be enough in a safe environment, adding predators to the game means that the prey control algorithm will need to follow a strategy to avoid them. Hence, the next subsections discuss strategies that the prey swarm can use to avoid the predator swarm.

Jump Strategy

The first strategy that this paper proposes is the jump strategy. This is a very basic strategy that involves making the prey agent 'jump' in a random direction, whenever a predator agent is in its proximity. Hence, the 'jumping force' can be defined as in equation 32.

$$\vec{j_i} = \begin{cases} \text{random\_dir}(), & \text{if} \|\vec{p_k} - \vec{p_i}\| \leq p\_d \text{ for some } k \in P \\ 0, & \text{otherwise} \end{cases} \tag{32}$$

$$\begin{aligned} \text{where:} \quad P &= \text{the set of all predators} \\ p\_d &= \text{the prey's perception distance} \\ \vec{r_i} &= \text{the position of the prey agent} \\ \vec{r_k} &= \text{the position of some predator agent} \end{aligned}$$

Integrating the jump strategy into the boid model is, then, as simple as adding the 'jumping force' as a weighted term to the resulting steering force, as shown in equation 35.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m_i} + \text{C.A.} + T(\vec{t} - \vec{p_i}) + J\vec{j_i} \tag{33}$$

Evasive Strategy

The evasive strategy builds upon the jump strategy. Rather than 'jumping' in a random direction, the evasive strategy proposes that the prey agent moves away from all of the predator agents in its proximity. To confirm to this, we will define the corresponding force as the sum of all relative position vectors between the agent and the predators in its proximity, as represented by equation 34

$$\vec{e_i} = \sum_{k\in N_p} (\vec{p_i} - \vec{p_k}) \tag{34}$$

where: N_p = the set of all nearby predators

$\vec{p_i}$ = the position of the prey agent

$\vec{p_k}$ = the position of a nearby predator agent

Same as before, the integration of the evasive strategy in the model is simple and can be achieved by adding the evasive force as a weighted term.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m}_i + \text{C.A.} + T(\vec{t} - \vec{p_i}) + E\vec{e_i} \tag{35}$$

Split Strategy

In contrast to the evasive strategy, the split strategy tries to confuse the predators by splitting the swarm into 2. Without an advanced control algorithm that tackles this case, the predator swarm will chase only one of the 2 sub-swarms, or it might even fail to chase any of them.

The 2 sub-swarms are obtained based on the parity of the agent id. Even ids become part of the first sub-swarm, while odd ids are part of the second sub-swarm. An anti-neighbour force is also introduced. The anti-neighbour force has the role of pushing away neighbours that are part of different sub-swarms, with the purpose of facilitating the physical separation of the 2 swarms.

We will start by defining $\vec{ca}_i$, which represents the mean position of the neighbours of agent i that are part of a different subswarm.

$$\vec{ca}_i = \frac{1}{|A_i|} \sum_{j \in A_i} \vec{p}_j \tag{36}$$

Based on the center-position, we define the anti-neighbor force $\vec{a}_i$ as in equation 37.

$$\vec{a}_i = \vec{p}_i - \vec{ca}_i \tag{37}$$

The anti-neighbour force is introduced in a similar fashion as before.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m}_i + \text{C.A.} + T(\vec{t} - \vec{r_i}) + A\vec{a_i} \tag{38}$$

Note that the anti-neighbour force is only applied for a limited time Δt until the swarms are fully separated.

Explode Strategy

The explode strategy takes a different approach from all the other strategies above. The idea behind this strategy is to make the swarm 'explode', i.e. increase the distance in-between the swarm members, whenever the predators approach the prey swarm. Unlike the other strategies, the explode strategy requires the prey swarm to have communication capabilities since each prey agent needs to be able to inform the other agents in its swarm whenever a predator is detected.

The explosive force is triggered within the entire swarm, whenever one of the prey agents detects a predator. The force acts on each of the swarm agents from the direction of the swarm center, as illustrated by equation 39.

$$\vec{X}_i = \frac{\vec{p_i} - \vec{c}}{||\vec{p_i} - \vec{c}||^2} \tag{39}$$

The explosive force only acts for a pre-defined period of time Δt after it was triggered. When the time expires, the explosive force is simply removed from the boid model. As for all of the other forces, extending the boid model with the explode strategy reduces to adding the explosive force as a weighted term to the resulting steering force.

6.3 Predator Control Algorithm

The last direction in which the boids model is extended is predator control. The purpose of this family of control algorithms is to enable the predators to hunt their prey efficiently and, thus, prevent the prey swarm from reaching its objectives. In order to create an efficient predator control algorithm we need to create a good strategy that enables the predator swarm to intercept and hunt down as many members as possible from the prey swarm. In light of this, the following subsections will explore different strategies that can be used to achieve this.

Center Steering Strategy

The first and the most simple predator strategy is the center steering strategy. This strategy consists of getting all of the predator agents to move towards the center of the prey swarm. This is implemented in a similar fashion to the way the prey control algorithm is made to gravitate towards the target area. The formula of the steering force is expressed in equation 52.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m}_i + \text{C.A.} + T\frac{1}{|B|}\sum_{j \in B}(\vec{p}_j - \vec{p}_i) \tag{40}$$

Proportional Navigation Strategy

An improvement over the center steering strategy is the proportional navigation strategy. This strategy is inspired by the well-known PNG law used by missiles to intercept their targets [13], but it is adapted to the boid's computational model.

The idea behind this strategy is to predict where the prey swarm is heading and, based on that, adjust the direction of the predator swarm in order to intercept it. For this strategy to be applicable, the predator swarm needs to be able to detect the instant velocity of each of the prey.

While the traditional PNG law outputs the acceleration that should be applied to the pursuer in order to intercept the target, the strategy proposed by this paper outputs the velocity $\vec{v}_p$ that each predator boid should aim to have in order to intercept the prey.

The velocity law is derived mathematically from the collision pyramid (see figure 5), which is a geometrical structure that indicates where 2 point-masses would collide assuming that they maintain their current velocity. Since none of the swarms maintain their velocities, the velocity law is used on every timestamp and the resulting velocity is integrated into the predator boid's steering force.

We will now go on to derive the mathematics behind this strategy. The first aspect that we need to observe is that in

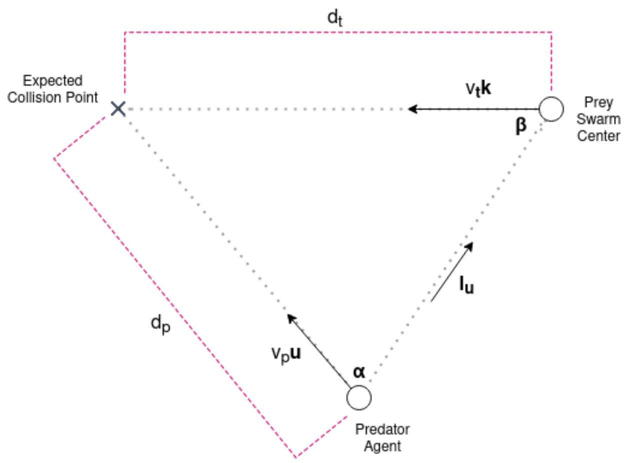


Figure 5: Collision pyramid.

order for the predator agent to collide with the prey swarm center, both entities need to reach the expected collision point at the same time. Thus, we can write the following:

$$v_p = \frac{d_p}{\Delta t} \iff \Delta t = \frac{d_p}{v_p} \tag{41}$$

$$v_t = \frac{d_t}{\Delta t} \iff \Delta t = \frac{d_t}{v_t} \tag{42}$$

$$\frac{d_p}{v_p} = \frac{d_t}{v_t} \tag{43}$$

Using the law of sines [9], we can express the angle α between the direction of the predator and the line of sight (i.e. the direction of the prey from the perspective of the predator) in the following way:

$$\frac{d_t}{\sin(\alpha)} = \frac{d_p}{\sin(\beta)} = \frac{d_p}{\|\vec{k} \times \vec{l}_u\|} \tag{44}$$

$$\sin(\alpha) = \frac{d_t}{d_p}\|\vec{k} \times \vec{l}_u\| = \frac{v_t}{v_p}\|\vec{k} \times \vec{l}_u\| \tag{45}$$

The unit vector $\vec{u}$ defining the direction that the predator agent needs to follow in order to intercept the prey can be found by rotating the unit vector $\vec{l}_u$ defining the line of sight by α, which is expressed mathematically as follows.

$$\vec{u} = R(\alpha)\vec{l}_u = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \vec{l}_u = \tag{46}$$

$$= \begin{bmatrix} \sqrt{1-\sin(\alpha)^2} & -\sin(\alpha) \\ \sin(\alpha) & \sqrt{1-\sin(\alpha)^2} \end{bmatrix} \vec{l}_u \tag{47}$$

The next step consists of determining the velocities for the predator agent and the prey swarm center. The speed of the predator agent is constant and chosen as a parameter for the algorithm. We express this mathematically as $v_p\vec{u}$, where $v_p =$ const. The velocity of the prey swarm center is more complicated, however. Since the prey swarm center is an aggregated entity, its corresponding velocity needs to be approximated from the velocities of each of the swarm members. Ideally, we would like to compute an approximation that does not change trajectory too much, in order to obtain smoother behaviour. It is known that the prey swarm will be seeking to reach the target area. Thus, the approximation will be computed by projecting the average velocity of the swarm in the direction of the target area. Mathematically, this approximation is presented in equation 48. Based on the approximated velocity vector, the absolute speed v_t of the prey swarm center and the direction unit vector $\vec{k}$ can be obtained, as shown in equation 49.

$$\vec{v_t} = (\frac{1}{|B|}\sum_{i \in B} \vec{v}_i)\frac{\vec{t} - \frac{1}{|B|}\sum_{i \in B}\vec{p}_i}{\|\vec{t} - \frac{1}{|B|}\sum_{i \in B}\vec{p}_i\|} \tag{48}$$

$$v_t = \|\vec{v_t}\|, \vec{k} = \frac{\vec{v_t}}{\|\vec{v_t}\|} \tag{49}$$

Now that all relevant equations are laied out, we can put together the control algorithm. The speed of the predator v_p is a pre-configured constant. The approximated speed of the prey swarm v_t can be derived from equation 48, while the unit vector $\vec{k}$ indicating the direction of the prey swarm can be derived from equation 49. The line of sight vector $\vec{l}_u$ can easily be determined by subtracting the relevant position vectors and normalizing the result. Based on these quantities, as well as equation 47, the unit vector $\vec{u}$ indicating the direction towards which the predator needs to travel can be computed as follows.

$$c_u = \frac{v_t}{v_p}\|\vec{k} \times \vec{l}_u\| \tag{50}$$

$$\vec{u} = \begin{bmatrix} \sqrt{1-c_u^2} & -c_u \\ c_u & \sqrt{1-c_u^2} \end{bmatrix} \vec{l}_u \tag{51}$$

The last step we need to look at is how to integrate everything into the existing boids model. In a similar fashion to all other algorithms presented in this paper, extending the boids model with the proportional navigation strategy boils down to adding the desired predator velocity $v_p\vec{u}$ as a weighted term to the boid's steering force.

$$\vec{f_i} = S\vec{s_i} + K\vec{k_i} + M\vec{m}_i + \text{C.A.} + Tv_p\vec{u} \tag{52}$$

Clustering Strategy

The last proposed strategy for the predator swarm is the clustering strategy. This kind of strategy arises from the need to enable the predator swarm to counter the split and explode strategies, that split the prey swarm into sub-swarms.

The idea behind the clustering strategy is to make use of a clustering algorithm to identify sub-swarms and then act accordingly, by splitting the predator swarm in a way that enables chasing the sub-swarms simultaneously. The clustering algorithm that this paper proposes is K-Means [14]. While K-Means enables us to find which prey belongs to each subswarm (a.k.a. cluster), there is still no way to determine how many sub-swarms / clusters to expect.

The standard way to approach the problem of finding the ideal number of clusters for K-Means is a method called 'knee point detection' [17]. The method involves running K-Means for different values of k and, then, computing the SSE (sum of squared errors) for each of the centroids determined by K-Means. The 'knee point' or 'the inflexion point' is represented by the value of k after which, if we increase k, there will be only a small drop in SSE. One popular algorithm developed to solve this problem is 'Kneedle', and it is what the proposed implementation uses [12]. Multiple clustering algorithms could theoretically be used, but exploring all of them is beyond the scope of this research.

7 Experimental Results

As presented in the methodology section, all algorithms proposed in this paper were implemented and run within a self-developed simulator/simulation framework that allows for the creation of benchmarks. Various types of experiments were run, in order to gain more insight into how the proposed algorithms work. The setup and results of those experiments are presented in the following sections.

The 2 benchmarked characteristics of the algorithms were the success rate and the average trial time. The success rate represents the average percentage of prey agents that managed to reach their target, while the average trial time measures how long it takes on average for all the prey agents to get to the target. Hence, a higher success rate and a lower average time indicate that the prey was successful. In contrast, a lower success rate and a higher average time mean a more successful predator, given a competitive scenario.

It is important to take into account that the outcomes of the experiments were to a large extent influenced by the chosen hyper-parameters for all the different algorithms. Finer tuning the hyper-parameters of some of the algorithms could potentially lead to better performances, however, this is left for further research. The exact hyper-parameters that were used in the experiments can be found in the appendix.

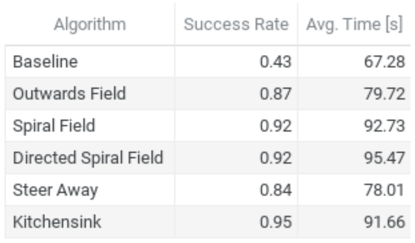
7.1 Obstacle Avoidance Experiment

The purpose of this experiment was to explore how the different proposed obstacle avoidance algorithms compare with each other. Since for this experiment predator-prey interactions were of no interest, the testing environment consisted of only a prey swarm trying to reach the target area.

The algorithms that were benchmarked in this experiment were the following: *baseline* (prey swarm without obstacle avoidance, used as a reference point), *outwards field*, *spiral field*, *directional spiral field*, *steer away*, *kitchensink* (steer away + directed spiral field). The benchmark was performed in both environments with 10 obstacles (figure 6) and 30 obstacles (figure 7).

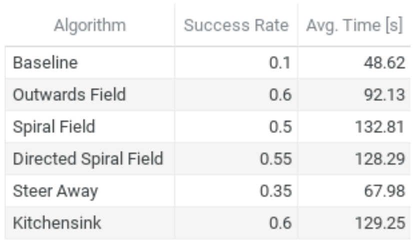
All algorithms perform better than the baseline, which means that all of them succeed at achieving obstacle avoidance to a certain extent. The best 2 performing algorithms seem to be the classical outwards field and the kitchen sink approach, however, the classical outwards field is somewhat faster.

It is interesting to notice that the spiral field algorithms decrease drastically in performance as the number of obstacles increases. From empirical observations, a likely explanation is that situations with many 'spiral field obstacles' around the agent tend to create a sort of 'vortex deadlocks', trapping the agent inside them. This would require further analysis, since being able to identify those deadlocks could boost the performance of this class of algorithms.



Algorithm	Success Rate	Avg. Time [s]
Baseline	0.43	67.28
Outwards Field	0.87	79.72
Spiral Field	0.92	92.73
Directed Spiral Field	0.92	95.47
Steer Away	0.84	78.01
Kitchensink	0.95	91.66

Figure 6: Obstacle avoidance benchmark. Results averaged over 1000 trials in environments with 10 random obstacles.



Algorithm	Success Rate	Avg. Time [s]
Baseline	0.1	48.62
Outwards Field	0.6	92.13
Spiral Field	0.5	132.81
Directed Spiral Field	0.55	128.29
Steer Away	0.35	67.98
Kitchensink	0.6	129.25

Figure 7: Obstacle avoidance benchmark. Results averaged over 1000 trials in environments with 30 random obstacles.

7.2 Prey vs Predator Experiment

This experiment was meant to analyse how different prey and predator control algorithms interact with each other, in order to gain insight into their performance and behaviour.

The prey algorithms that took part in the benchmark were the following: *baseline* (prey swarm with obstacle avoidance but no predator evasion strategy), *evasive*, *explode*, *explode-evasive* (combination of explode and evasive), jump, *split*, *split-evasive* (combination of split and evasive). All prey algorithms used the kitchensink approach for obstacle avoidance.

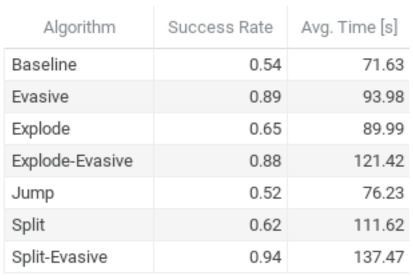
In contrast, the predator algorithms that were benchmarked were the *center steering strategy* and the proportional navigation strategy. All predator algorithms used the outwards field obstacle avoidance.

The benchmarks were run on 1000 trials, in both situations with no obstacles (see figures 8, 10) and situations with 10 randomly positioned obstacles (see figures 9, 11).

The first aspect to notice from this experiment is that, except for the jump strategy, all other strategies performed better than the baseline. This means that, except for the jump strategy, all other prey control algorithms are viable solutions. The jump strategy was not the best heuristic and it was only chosen as a starting point, so the result makes sense.

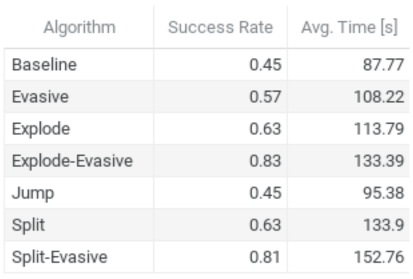
Out of all prey control algorithms, the explode-evasive and the split-evasive strategies seem to have the highest success rates across all test cases. This is consistent with the theory, since neither the center-steering algorithm nor the proportional navigation algorithm has a good way to deal with sub-swarms.

One other aspect to notice is that the evasive strategy seems to be particularly vulnerable to the proportional navigation strategy. The proportional navigation strategy, however, seems to perform worse in obstacle environments. This is again, expected since the obstacle interferes with the collision pyramid.



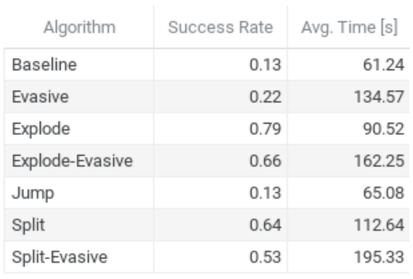
Algorithm	Success Rate	Avg. Time [s]
Baseline	0.54	71.63
Evasive	0.89	93.98
Explode	0.65	89.99
Explode-Evasive	0.88	121.42
Jump	0.52	76.23
Split	0.62	111.62
Split-Evasive	0.94	137.47

Figure 8: Prey control algorithms benchmarked against center steering predators. Results averaged over 1000 trials in environments without obstacles.



Algorithm	Success Rate	Avg. Time [s]
Baseline	0.45	87.77
Evasive	0.57	108.22
Explode	0.63	113.79
Explode-Evasive	0.83	133.39
Jump	0.45	95.38
Split	0.63	133.9
Split-Evasive	0.81	152.76

Figure 9: Prey control algorithms benchmarked against center steering predators. Results averaged over 1000 trials in environments with 10 random obstacles.

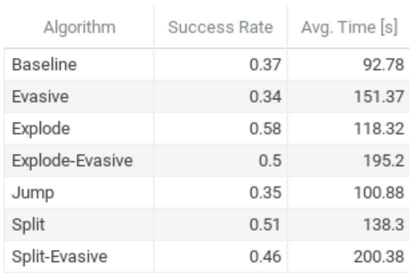


Algorithm	Success Rate	Avg. Time [s]
Baseline	0.13	61.24
Evasive	0.22	134.57
Explode	0.79	90.52
Explode-Evasive	0.66	162.25
Jump	0.13	65.08
Split	0.64	112.64
Split-Evasive	0.53	195.33

Figure 10: Prey control algorithms benchmarked against proportional navigation predators. Results averaged over 1000 trials in a environments without obstacles.

7.3 Cluster Strategy

The cluster strategy proved too computationally expensive for any relevant benchmarks to be performed. A less computationally intensive way of clustering should be researched in order to facilitate experiments.



Algorithm	Success Rate	Avg. Time [s]
Baseline	0.37	92.78
Evasive	0.34	151.37
Explode	0.58	118.32
Explode-Evasive	0.5	195.2
Jump	0.35	100.88
Split	0.51	138.3
Split-Evasive	0.46	200.38

Figure 11: Prey control algorithms benchmarked against proportional navigation predators. Results averaged over 1000 trials in environments with 10 random obstacles.

8 Responsible Research

As for any research project the reproducibility of the experiments is very important in order to facilitate peer review and allow others to make sure that the conclusions derived as a result of the research are valid. While this paper conceptually explains all algorithms that were used in the experiments, as well as the mathematics that goes behind them, implementation details are not discussed. On top of that, certain details with regard to how the simulation environment works, such as the obstacle field generator, are also not tackled in this paper, due to them not being central to the research questions at hand. All of those factors lead to a situation in which, based only on this paper, the results might not be reproducible.

In order to tackle this issue, the simulated environment that was used for the experiments, together with all of the algorithm implementations were released to the public in an open source manner. In this way, the reader of the paper can look over the code, in order to see how the algorithms were exactly implemented. The user can also use the simulated environment to test his own algorithms and, possibly, perform further research. A link to the repository that contains the simulated environment can be found in the appendix.

9 Conclusions and Future Work

The goal of this research was to explore various swarm control algorithms in a target-oriented predator-prey situation with an obstacle-filled environment. The theoretical background that was chosen for the research was the boids computational model. In light of this, the research was split into 2 parts.

The first part saw the creation, exploration and benchmark of obstacle avoidance algorithms, common for both predator and prey situations. All algorithms performed better that the baseline (which consisted of no obstacle avoidance), which means that the proposed algorithms worked. The efficiency of the algorithms could be further improved if hyper-parameter optimization was performed, which can be a future research direction. On top of that, a mathematical analysis of deadlock situations (situations in which the agent gets stuck) could lead to more promising results, especially when it comes to the spiral field algorithm.

The second part of the research dealt with the arms race style development, exploration and benchmarking of prey and predator algorithms. Most proposed algorithms performed better than the baseline. The jump prey heuristic turned out not to work, and I believe that no more effort should be spent in its direction. The cluster predator strategy was too computationally intensive to benchmark. The creation of a cluster-based predator that is easier to simulate would represent an interesting continuation of the project. As in the case of obstacle avoidance, hyper-parameter optimization should be performed to boost the capabilities of the proposed algorithms. Neural network based approaches to both the prey swarms and the predator swarms could also be promising continuations of the research.

A Problem Formalization

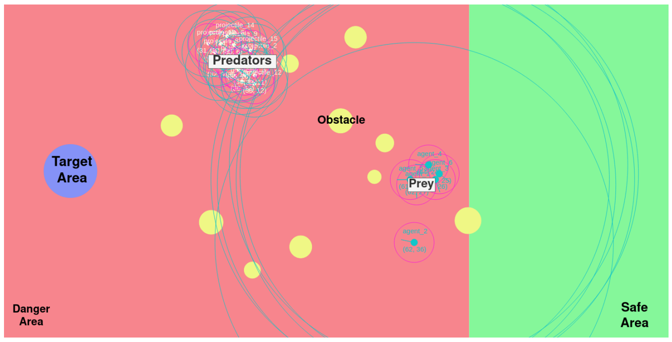


Figure 12: Explanation of the problem environment

B Notation Explanations

B.1 Related Work

- F - force in the parallel spring-damper system [N]
- F_{spring} - force in the spring [N]
- F_{damper} - force in the damper [N]
- k - elastic constant [N/m]
- l - current elongation [m]
- l_0 - initial elongation [m]
- c - damping coefficient [N*s / m]
- $\vec{a}_i$ - target acceleration for the agent, according to the control algorithm [m/s$\hat{2}$]
- m_i - mass of agent i [kg]
- N_i - the neihbours of agent i
- $\vec{F}_{ij}$ - force in the damper between agent i and agent j

B.2 Theoretical Background

- b_i - boid i
- $\vec{p}_i$ - position vector of boid i
- $\vec{v}_i$ - velocity vector of boid i
- $\vec{N}_i$ - neighbors of boid i
- $\vec{c}_i$ - mean position of neighbors
- $\vec{s}_i$ - separation force of boid i
- $\vec{k}_i$ - cohesion force of boid i
- $\vec{m}_i$ - matching force of boid i
- $\vec{f}_i$ - steering force of boid i

B.3 Algorithm Development

- $S, K, M, A, T, E, G, H, J, X$ - real coefficients
- $\vec{g}(\vec{r})$ - outwards force field
- w_g - outwards field strength
- d - threshold distance in various contexts
- $\vec{h}(\vec{r})$ - spiral force field
- w_h - spiral force field outwards component strength
- $\vec{h}_a(\vec{r})$ - anti-clockwise spiral force field
- $\vec{h}_c(\vec{r})$ - clockwise spiral force field
- $\vec{h}_{dir}(\vec{r}, \vec{s})$ - directional force field
- $\vec{s}$ - steering direction (difference between the target's position vector and the boid's position vector)
- $\vec{t}$ - position vector of target area
- $\vec{o_j}$ - position vector of object j
- S_1 - the radius of the agent (see diagram)
- S_2 - the radius of the obstacle (see diagram)
- l_1 - distance between agent center and boundary intersection point (see diagram)
- l_2 - distance between obstacle center and boundary intersection point (see diagram)
- α - boundary angle (steer away context)
- β - actual steering angle with respect to obstacle (i.e. if β is within $+\alpha$ and $-\alpha$, the agent will hit the target (steer away context)
- $\Delta\vec{r}$ - relative position vector between agent and obstacle
- $R(x)$ - 2D rotational matrix
- γ - small steer away adjustment angle
- A_i - set of anti-neighbors of agent i
- C.A. - genetic collision avoidance term.
- $\vec{f}_i$ - steering force driving the agents.
- B - set of all prey boids.
- P - set of all predators.
- $\vec{e}_i$ - evasive force
- $\vec{ca}_i$ - mean position of neighbors of agent i that are part of a different sub-swarm.
- $\vec{a}_i$ - anti-neighbor force
- d_t - distance between target and expected collision point (in the context of the collision pyramid).
- d_p - distance between predator and expected collision point (in the context of the collision pyramid).
- v_p - predator / pursuer speed
- v_t - approximated target / prey speed

C Experiments hyper-parameter list

- M - 0.05
- K - 0.005
- S - 0.05
- w_g - 10
- G - 1.0
- H - 1.0
- T - 0.1
- w_h = 10
- E - 0.01
- X - 20.0
- explosion time - 50
- J - 3.0
- A - 200.0
- anti neighbor time - 30.0
- anti neighbor distance - 50.0
- perception distance - 30
- swarm distance - 3
- number of prey agents - 6
- number of predator agents - 15

D Simulator Repository

The simulator can be found on a public github repository at the following link *https://github.com/catalinlup/SwarmResearchSimulator*. More details about how to use it can be found in the README.md section.

References

[1] Peter A. Abrams. The evolution of predator-prey interactions: Theory and evidence. *Annual Review of Ecology and Systematics*, 31(1):79–105, 2000.

[2] Roland Bouffanais. *Design and Control of Swarm Dynamics*, chapter 2. Springer, 2016.

[3] Yen-Wei Chen, Kanami Kobayashi, Xinyin Huang, and Zensho Nakao. Genetic algorithms for optimization of boids model. In Bogdan Gabrys, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, pages 55–62, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[4] Carlos Delgado-Mata, Jesus Ibanez Martinez, Simon Bee, Rocio Ruiz-Rodarte, and Ruth Aylett. On the use of virtual animals with artificial fear in virtual environments. *New Generation Computing*, 25(2):145–169, Feb 2007.

[5] Faiza Gul, Wan Rahiman, and Syed Sahal Nazli Alhady. A comprehensive study for robot navigation techniques. *Cogent Engineering*, 6(1):1632046, 2019.

[6] Christopher Hartman and Bedrich Benes. Autonomous boids. *Computer Animation and Virtual Worlds*, 17, 2006.

[7] Adil Hashim, Tanya Saini, Hemant Bhardwaj, Adityan Jothi, and Ammannagari Vinay Kumar. *Application of Swarm Intelligence in Autonomous Cars for Obstacle Avoidance*, pages 393–404. Springer Singapore, Singapore, 2019.

[8] Keith Head. Gravity for beginners. *University of British Columbia*, 2053, 2003.

[9] PATRIK NYSTEDT. A proof of the law of sines using the law of cosines. *Mathematics Magazine*, 90(3):180–181, 2017.

[10] Sahithi Podila and Ying Zhu. Simulating a predator fish attacking a school of prey fish in 3d graphics. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Fatih Porikli, Sandra Skaff, Alireza Entezari, Jianyuan Min, Daisuke Iwai, Amela Sadagic, Carlos Scheidegger, and Tobias Isenberg, editors, *Advances in Visual Computing*, pages 586–594, Cham, 2016. Springer International Publishing.

[11] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, aug 1987.

[12] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, 2011.

[13] U.S. Shukla and P.R. Mahapatra. The proportional navigation dilemma-pure or true? *IEEE Transactions on Aerospace and Electronic Systems*, 26(2):382–392, 1990.

[14] Douglas. Steinley. K-means clustering: A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1):1–34, 2006.

[15] STM. Kargu - the kamikaze drones getting ready for the swarm operation. https://www.youtube.com/watch?v=3d28APIfwSI, 2019.

[16] Jakub Wiech, Victor A. Eremeyev, and Ivan Giorgio. Virtual spring damper method for nonholonomic robotic swarm self-organization and leader following. *Continuum Mechanics and Thermodynamics*, 30(5):1091–1102, Sep 2018.

[17] Qinpei Zhao, Ville Hautamaki, and Pasi Fränti. Knee point detection in bic for detecting the number of clusters. In Jacques Blanc-Talon, Salah Bourennane, Wilfried Philips, Dan Popescu, and Paul Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems*, pages 664–673, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.