Controlling a Swarm of Robots in a Simulator Stage using Reynolds' Rules

Joseph Adeola, Khawaja Ghulam Alamdar, Moses Ebere, Nada Elsayed Abbas

Abstract—This project focuses on implementing Craig Reynolds' behavioral rules for controlling a swarm of robots in a simulated environment. The implementation involves initially incorporating fundamental rules such as Separation, Alignment, and Cohesion, followed by the development of additional rules for Navigation to a specific point and Obstacle Avoidance. The robots, represented as omni-directional points in the Stage simulator, were tested in various scenarios to assess their behavior in different environments. The goal is to explore and understand the adaptability and performance of swarm behaviors using Reynolds' rules and additional algorithms in simulated settings. Simulation results are included here and here.

Index Terms—Swarm Robotics, Flocking, Reynolds, Biomimetics, Obstacle Avoidance

1 Introduction

Swarm robotics has become an increasingly significant domain in the field of robotics, drawing inspiration from the collective behaviors observed in natural systems such as flocks of birds, schools of fish, and colonies of insects. This burgeoning area seeks to understand, model, and replicate these intricate behaviors within artificial systems comprising multiple autonomous robots. Craig Reynolds' seminal work in 1987 introduced a set of behavioral rules, serving as fundamental principles for simulating emergent behaviors in decentralized systems.

The objective of this project is to delve into the practical implementation and assessment of Reynolds' behavioral rules within a simulated environment in a (ROS) framework and the Stage simulator. Building upon the three core rules of Reynolds'; separation, alignment, and cohesion, the project aims to extend the swarm's capabilities by introducing Navigation and Obstacle Avoidance behaviors. Navigation enables the swarm to move towards predefined points within the environment, gradually decelerating as they approach the target and coming to a stop at the designated location. Simultaneously, the Obstacle Avoidance rule equips the swarm with the capability to detect and evade obstacles present in the environment, adjusting their trajectories to circumvent potential collisions.

The simulated robots, represented as omnidirectional entities, undergo rigorous testing across various simulated scenarios and environments. The evaluation focuses on analyzing the swarm's collective behavior, adaptability, and robustness in response to diverse environmental conditions.

This project seeks to bridge theoretical concepts with practical implementation, offering insights into the dynamics of collective behaviors exhibited by robotic swarms governed by Reynolds' rules and additional algorithms within simulated settings. Video results are provided here and here.

2 Methodology

2.1 Reynolds' Basic Behaviors

Reynolds' basic behaviors in a boid simulation system comprise three fundamental principles:

1. **Cohesion**: Cohesion encourages boids to move towards the average position of neighboring boids within a defined spatial range. Mathematically, cohesion can be expressed as:

$$Cohesion = \frac{1}{N}\sum_{j=1}^{N}(\mathbf{x_j} - \mathbf{x_i})$$

where: N denotes the number of neighboring boids, x_i signifies the position of the current boid, and x_j represents the position of the j-th neighboring boid.

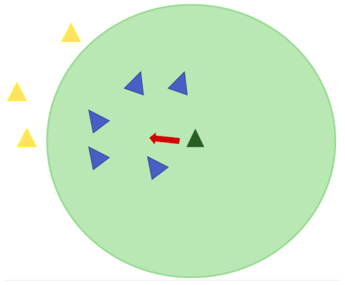


Fig. 1: Cohesion

2. **Separation**: Separation aims to avoid collisions by steering boids away from nearby boids within a specified minimum distance. The separation behavior can be mathematically represented as:

$$Separation = \frac{1}{N}\sum_{j=1}^{N}\frac{\mathbf{x_i} - \mathbf{x_j}}{\|\mathbf{x_i} - \mathbf{x_j}\|}$$

where the denominator denotes the Euclidean distance between the current boid x_i and its neighboring boids x_j.

3. **Alignment**: Alignment aligns the heading direction of a boid with the average heading direction of its neighbors. The alignment behavior can be expressed as:

$$Alignment = \frac{1}{N}\sum_{j=1}^{N}(\dot{\mathbf{x}}_\mathbf{j} - \dot{\mathbf{x}}_\mathbf{i})$$

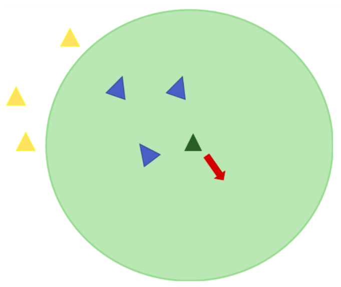


Fig. 2: Separation

where $\dot{x}_j$ represents the velocity (change in position) of the j-th neighboring boid.

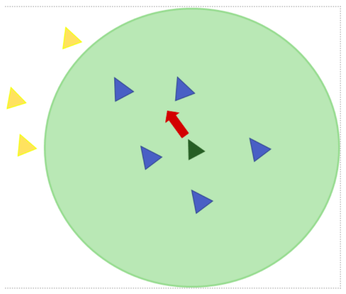


Fig. 3: Alignment

These behaviors, namely cohesion, separation, and alignment, serve as foundational principles governing the interactions and movements of boids within a simulated environment, influencing emergent collective behaviors.

2.2 Local Neighborhood

The identification of a local neighborhood within a boid simulation system involves determining neighboring agents based on spatial proximity and directional alignment. This mathematical framework serves as a foundation for characterizing neighboring boids relative to a focal agent.

1. **Spatial Proximity:**

The Euclidean distance ($distance$) between two boids positioned in a two-dimensional space is expressed as:

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

where (x_1, y_1) and (x_2, y_2) denote the positions of two boids.

2. **Directional Alignment:**

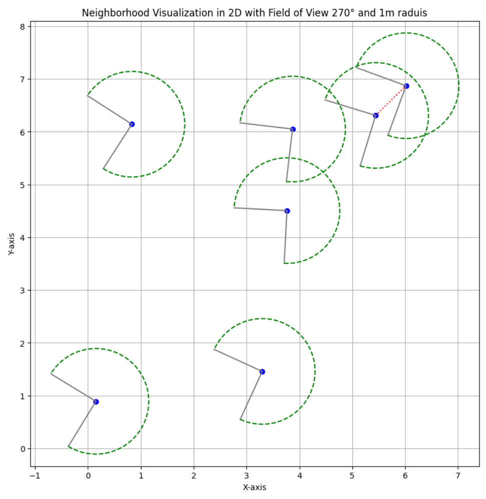
The angular disparity (β) between two boids, considering their positions (x_1, y_1) and (x_2, y_2), can be calculated using the `atan2` function:

$$\alpha = atan2(y_2 - y_1, x_2 - x_1)$$

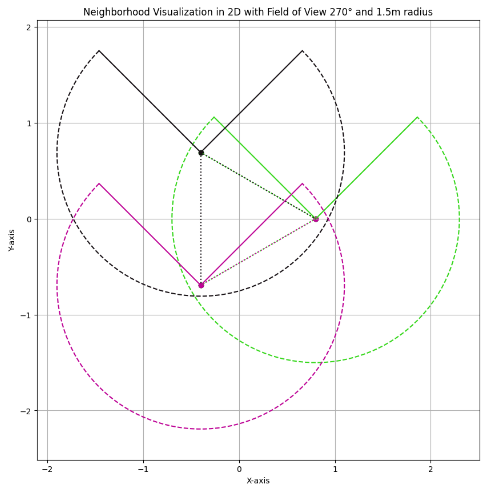
The calculation of angular difference is represented as:

$$\beta = \alpha - \theta$$

Where β is difference between agent(1) direction and Spatial Proximity direction between Agent(1)and Agent(2), α is Spatial Proximity direction between Agent(1)and Agent(2), and θ is the angle of Agent(1) direction then normalizing it within the ([- π, π])



(a) 10 boids - random formation



(b) 3 boids - uniform formation

Fig. 4: Local perception testing case

Changes in these angles will directly influence the resulting angular difference, affecting the formation of the local neighborhood. The criteria for identifying neighboring boids within a local neighborhood involve within the local perception distance and field of view.

$$(distance \leq r)\ and\ (|\beta| \leq \frac{FoV}{2})$$

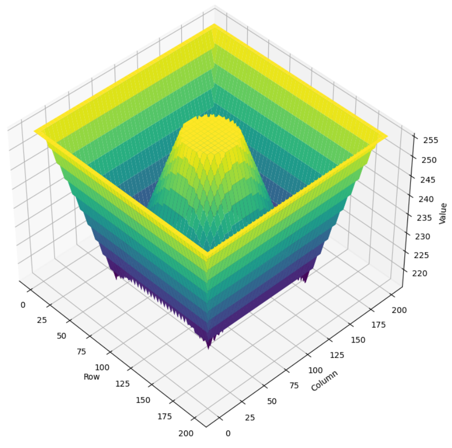
where r signifies the spatial radius and the FoV represents the angular width, influencing the definition and size of the local neighborhood.

2.3 Obstacle Avoidance

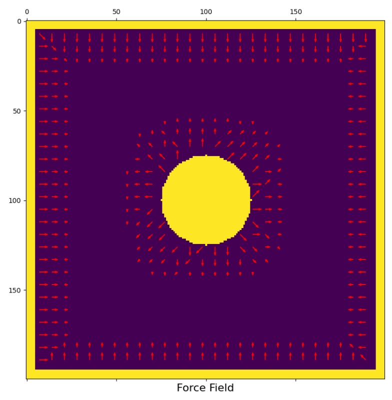
The behaviors described above ensure cohesive flock movement among the boids while preventing collisions with each other. However, the environment may introduce obstacles that necessitate avoidance maneuvers by the boids. In the following subsections, two strategies are presented to address this crucial aspect of behavioral response.

2.3.1 Potential Field

The Brushfire algorithm, also recognized as the wavefront planner and initially introduced in [1], serves the purpose



(a) Potential Field Function



(b) Force Field

Fig. 5: Potential Field Functions for A Bounded Circular Obstacle

of generating the potential function for the obstacles. This algorithm assigns values to individual cells within the grid, reflecting the distance from obstacles.

The potential function V, calculated for a circular object in a confined rectangular environment, is depicted in Figure 5a. The potential is elevated in close proximity to the obstacles and diminishes linearly with increasing distance from the obstacle. The boid must navigate downhill to maintain a safe distance from the obstacle.

The force field associated with this potential function can be computed by determining its gradient:

$$\mathbf{F}[r,c] = \nabla V[r,c]$$

where r and c denote the grid map location.

Therefore, the repulsion direction exerted by an obstacle on the boid, expressed as a vector at any cell location, can be determined using this force field. The gradient derived provides only the direction. To represent the magnitude of the force vector, it is proportionally scaled based on the proximity of that cell to the obstacle. This proximity information is encoded in the output of the brushfire algorithm.

An illustration of the force field is presented in Figure 5b. As observed, the force vector extends orthogonally from the obstacles and exhibits a greater magnitude at cells closer to the obstacles. In practical terms, the potential function is confined to a specific radius around the obstacles, and consequently, so is the force field. This ensures that the boids only experience the influence of the force field when they are in close proximity to an obstacle.

Given that the environments in question are static, the potential field proves to be a computationally economical approach to obstacle avoidance. The field only needs to be generated at the onset of the simulation and saved as a 2D array $F(r,c)$, where the value at each cell location corresponds to the force in the x and y directions. These vectors are normalized within the range [0,1], facilitating easier tuning and integration with other behaviors. Consequently, during the simulation, each boid retrieves the force it should experience from the F array and combines it with other behaviors in the specified manner. This renders obstacle avoidance a computationally inexpensive process, involving just the retrieval of data from an array.

2.3.2 Steer to Avoid

The steer-to-avoid method orchestrates obstacle avoidance by assessing potential paths from a lookahead point, a calculated position determined by incrementally adjusting the orientation of an object. This lookahead point represents a hypothetical future position based on the object's current orientation and its potential angular adjustments.

By gradually altering the orientation angle in incremental steps, the algorithm systematically explores two directions: a rightward adjustment and a leftward adjustment from the current orientation. For each direction, it calculates a new orientation and subsequently derives a lookahead path that extends from the object's current position.

These calculated paths are crucial, serving as the basis for assessing their validity. The algorithm evaluates whether these paths encounter any obstacles or impediments in the environment. If the lookahead path resulting from a particular orientation adjustment is free from obstacles, the algorithm updates the object's orientation to align with this obstacle-free path, steering the object in a direction that avoids potential obstacles.

The process repeats iteratively until a valid, obstacle-free path is discovered or until the angular adjustments reach a limit without finding a clear path. At this point, the algorithm terminates, signaling the unavailability of a viable path that avoids obstacles within the given constraints.

Algorithm 1 Steer-to-avoid Obstacle Avoidance

```
while not path_valid do $\alpha$ += $step\_angle$
$newtheta = \theta v + \alpha$
Calculate the lookahead path based on $newtheta$
Check if the new path is valid
    if path is valid then Update the orientation & exit loop
$newtheta = \theta v - \alpha$
Calculate the lookahead path based on $newtheta$
Check if the new path is valid
        if path is valid then Update the orientation & exit
return boid_theta
```

Having the steer-to-avoid behavior as a part of the system, many forces could interfere with each other. The following figure shows a case where the boid adjusted its direction to avoid an obstacle but got pushed back by another

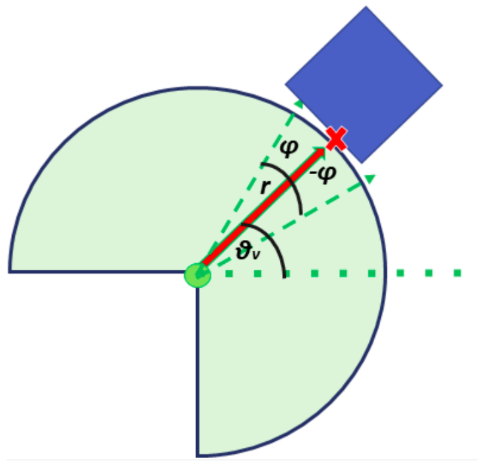


Fig. 6: Steer-to-avoid Diagram

force that can be a separation force, for example. This corner case was the reason why the method had to check the whole path ahead instead of only the look-ahead point. Because the path checking occurs in stepwise increments along the line, it also addresses the unavailability of the look-ahead point, such as cases when it lies outside the map boundaries as shown in Figure 7.

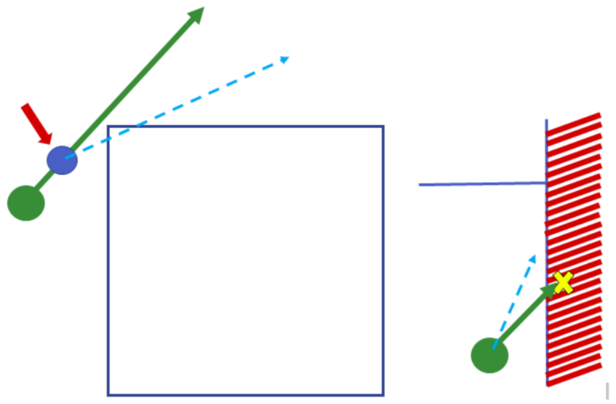


Fig. 7: Lookahead Point vs Lookahead Path

Algorithm 2 Check Path Function

```
dist ← CalculateDistance (p1, p2)
num_steps ← dist / step_size
num_steps ← integer_value(num_steps)
for j ← 0 num_steps - 1 do interpolation ← j / num_steps
x ← p1[0] × (1 - interpolation) + p2[0] × interpolation
y ← p1[1] × (1 - interpolation) + p2[1] × interpolation
waypoints.append((x, y))
waypoints ← waypoints[1:]
    for w in waypoints do
        if IsValid(w) is False then False
True
```

2.4 Seek

In the *Seek* behavior, the objective is to propel the flock towards a target at maximum speed. This is achieved by first calculating the vector offset between the robot's current position and the target location. The desired velocity vector is then derived by normalizing this offset and scaling it to the robot's maximum speed, enabling swift movement towards the target.

Algorithm 3 Seek Behavior

```
$desired\_velocity \leftarrow \text{NORMALIZE}(position - target) \times max\_speed$
$steering \leftarrow desired\_velocity - velocity$
```

2.5 Arrival

The *Arrival* behavior shares similarities with Seek in directing the flock toward the target from afar. However, it adds a critical refinement for the final approach: as the swarm gets closer to the target, this behavior carefully adjusts its speed, gradually slowing down until it comes to a complete stop at the target location. This process starts when the flock is still a considerable distance from the target, moving at its maximum speed. As it crosses into a predefined slowing distance, the speed begins to diminish. Beyond this radius, the desired velocity stays at the maximum limit, but once inside, it's steadily reduced to zero. To smoothen this convergence, the hyperbolic tangent function $(tanh)$ is used. This step-by-step reduction in speed, triggered when the current speed surpasses a set slowdown level and the swarm is within the slowing radius, ensures that the flock decelerates smoothly, ultimately stopping precisely at the goal. The steering force also decreases accordingly, reducing to zero as the target is neared.

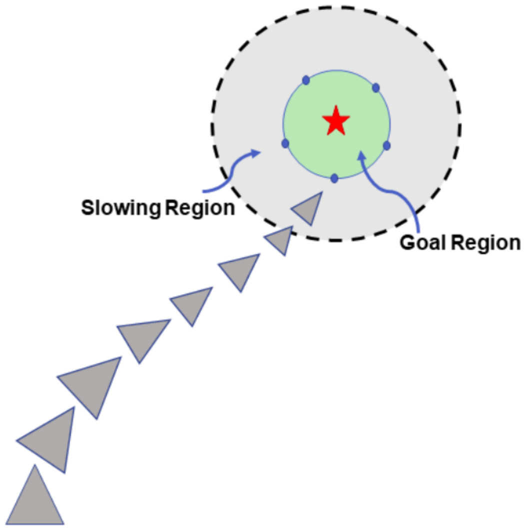


Fig. 8: Arrival

In our method, we conceptualize the goal as a circular area, centered at a designated point on the map, with a radius equal to the desired minimum separation distance. This circle serves as the target area for the flock. To effectively guide the flock, certain boids are selected as leaders. These leaders utilize the arrival steering force to steer the rest of the flock towards the goal, while the flock adheres to basic behavioral rules of separation, alignment, and cohesion.

Algorithm 4 Arrival Steering Behavior

```
$target\_offset \leftarrow target - position$
$distance \leftarrow \text{LENGTH}(target\_offset)$
$ramped\_speed \leftarrow max\_speed \times \left(\frac{distance}{slowing\_distance}\right)$
$clipped\_speed \leftarrow \text{MIN}(ramped\_speed, max\_speed)$
$desired\_velocity \leftarrow \left(\frac{clipped\_speed}{distance}\right) \times target\_offset$
$steering \leftarrow desired\_velocity$
```

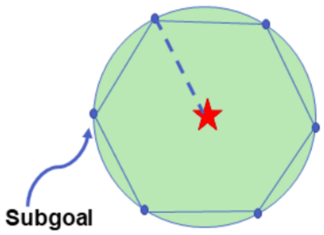


Fig. 9: Goal and Subgoals

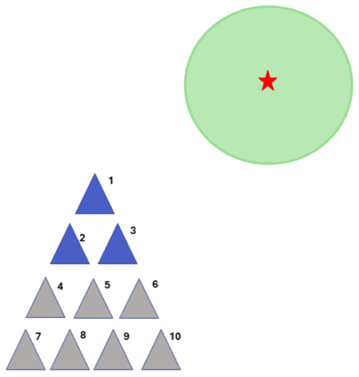
Leadership is assigned through the generation of several subgoals, corresponding to the number of desired leaders. These subgoals are equidistant points on the circumference of the target circle, spaced apart by a distance equal to the radius of the circle. This spacing aligns with the minimum desired separation distance between the boids.
We designed two methods for assigning leadership:

1) **Nearest Neighbors (NN):** Initially, the nearest boid to each subgoal is appointed as a leader. As the flock moves toward the goal and the formation changes, the leader for each subgoal is updated to be the closest boid.
2) **Convex Hull Method:** This approach considers only boids on the flock's periphery, with clear vision, as potential leaders. After identifying all boundary boids, those nearest to the subgoals are selected as leaders. Leadership is dynamically adjusted as the flock advances.

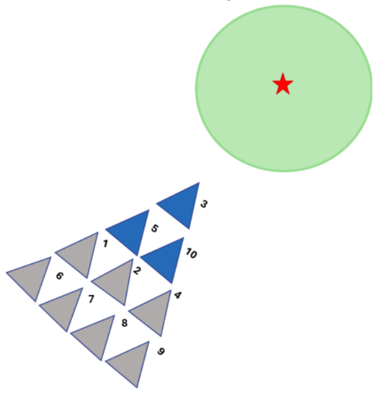
A boid is considered to have reached its destination once it enters the target area or reaches its boundary. This marks the completion of the arrival process as per our algorithm.

Algorithm 5 Arrival Behavior Implementation Steps

```
1: Define a goal as a circle with a radius of minimum separation distance at a specific map location.
2: Select leaders to guide the flock towards the goal.
3: Generate subgoals on the circle's circumference, equal to the number of desired leaders.
4: Apply one of the following methods to assign leaders:
5:     a. Nearest Neighbors (NN): Assign the closest boid to each subgoal as a leader, recalculating as the formation changes.
6:     b. Convex Hull Method: Identify boundary boids, then select those nearest to the subgoals as leaders, with dynamic reassessment as the flock moves. =0
```



(a) Nearest Neighbours



(b) New Leaders

Fig. 10: Dynamic Leader Selection

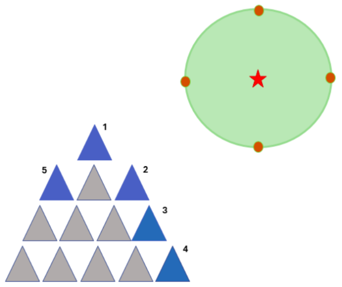


Fig. 11: Convex Hull

2.6 Prioritized Acceleration Allocation

Prioritized acceleration allocation operates on a hierarchy of behaviors, assigning acceleration requests based on their importance, which can shift in response to changing scenarios. Acceleration is allocated to behaviors according to a predetermined priority list.

Requests for acceleration are processed in this order until the cumulative demand surpasses the boid's maximum acceleration limit. Should this occur, the allocation for the last behavior is reduced to remain within the maximum limit. The navigation system ensures that acceleration is distributed first to address the most critical behaviors, potentially deferring less critical ones. For example, avoidance maneuvers to circumvent obstacles would take precedence over flock cohesion.

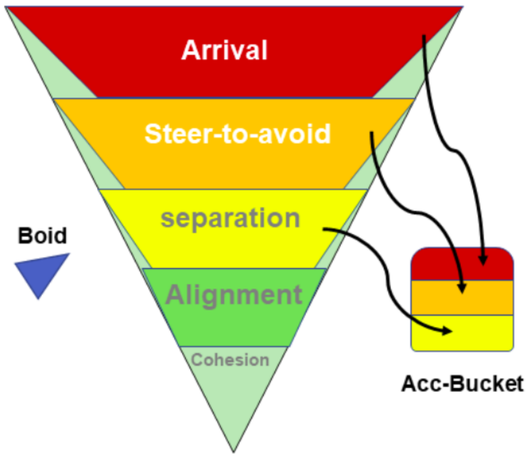


Fig. 12: Priority Pyramid

Algorithm 6 Prioritized Acceleration Allocation

```
1:  Initialize $totalAcceleration$ to zero vector
2:  Initialize $magnitudeAccumulator$ to zero
3:  Define $maxAcceleration$ for the boid
4:  for each behavior in $priorityList$ do
5:      $request \leftarrow$ ComputeAccelerationRequest(behavior)
6:      $magnitude \leftarrow$ ComputeMagnitude($request$)
7:      if $magnitudeAccumulator + magnitude > maxAcceleration$ then
8:          $request \leftarrow$ TrimRequest($request$, $maxAcceleration - magnitudeAccumulator$)
9:          $magnitude \leftarrow$ ComputeMagnitude($request$)
10:     end if
11:     $totalAcceleration \leftarrow totalAcceleration + request$
12:     $magnitudeAccumulator \leftarrow magnitudeAccumulator + magnitude$
13:     if $magnitudeAccumulator \geq maxAcceleration$ then
14:         break
15:     end if
16: end for
17: return $totalAcceleration$
```

3 Implementation

3.1 Software Architecture

The system's architecture is constructed upon the Robot Operating System (ROS), which facilitates the integration and processing of various data streams critical to autonomous navigation. Config Parameters initialize the system by defining operational constraints such as field of view (FOV) range and angle, behavior weights, priority lists, in-case of arrival (method for leader determination, and number of leaders), which are essential for delineating the interactive space between robots.

The determination of $Neighbours$ relies on the position and velocity information provided by ROS, filtered through the specified Config Parameters. This process ensures that only relevant nearby entities are considered for subsequent behavioral calculations.

$Goals$ input conveys target locations to the boids, directing navigational behaviors. The 'Seek' behavior propels the flock toward these targets, while the 'Arrival' behavior manages deceleration and precise stopping at the destination points.

The $Gridmap$ input supplies the layout of the environment, identifying navigable versus occupied spaces. This information is crucial for the 'Obstacle Avoidance' behavior, enabling the flock to navigate around impediments effectively.

The $Accumulator$ balances the output from the 'Basic Behaviours' module, which comprises 'Seek', 'Arrival', and 'Obstacle Avoidance'.Whether to employ a priority list is a configurable parameter that influences how the accumulator behaves. It prioritizes (if specified) and sums these behaviors to generate appropriate 'acceleration', scaling them based on the specified weight for each behavior. This prioritization is essential in scenarios where immediate environmental responses must be balanced with long-term navigational goals.

The $MotionModel$ calculates the new velocity $v = u+at$ by adding the acceleration from the $Accumulator$ with the boids prior velocity. It then issues $VelocityCommands$ that actuate the boids movement.

Finally, the $StageSimulator$ component simulates a physical environment for the robots, providing a feedback loop that refines the 'Motion Model' through 'Velocity Commands' derived from behavioral outputs. This simulation ensures that modeled behaviors are reflective of potential real-world scenarios. The developed system architecture is included in Figure 13.

4 Results and Discussion

4.1 Preliminary Behaviors and Navigation

Within the context of this simulation project, the developed algorithms were sufficiently tested in 6 map environments. These tests were run with different behavior combinations and parameter settings to ascertain the performance of algorithms. In this subsection, we examine some simulation results.

Run	Alignment	Cohesion	Separation
a	✓	×	×
b	×	✓	×
c	×	×	✓

TABLE 1: Simulation with Isolated Behaviors (for fig. 14)

The basic behaviors - separation, alignment, and cohesion - were tested in isolation in the "map with a frame" as indicated in Table 1. In the results given in Figure 14, we observe that while separation and cohesion act in opposite directions, alignment is only concerned with ensuring that boid i is moving at the same velocity as its neighbors. This is evident in the equal trails of displacement in Figure 14(a). When acting almost in isolation, separation moves boid i clear of its neighbors and stops sending acceleration requests. Additionally, cohesion will lead to a collision at the neighborhood centroid if acting alone. With all three behaviors acting on the boids, they eventually converge to a formation dictated by a balance of weights across the board.

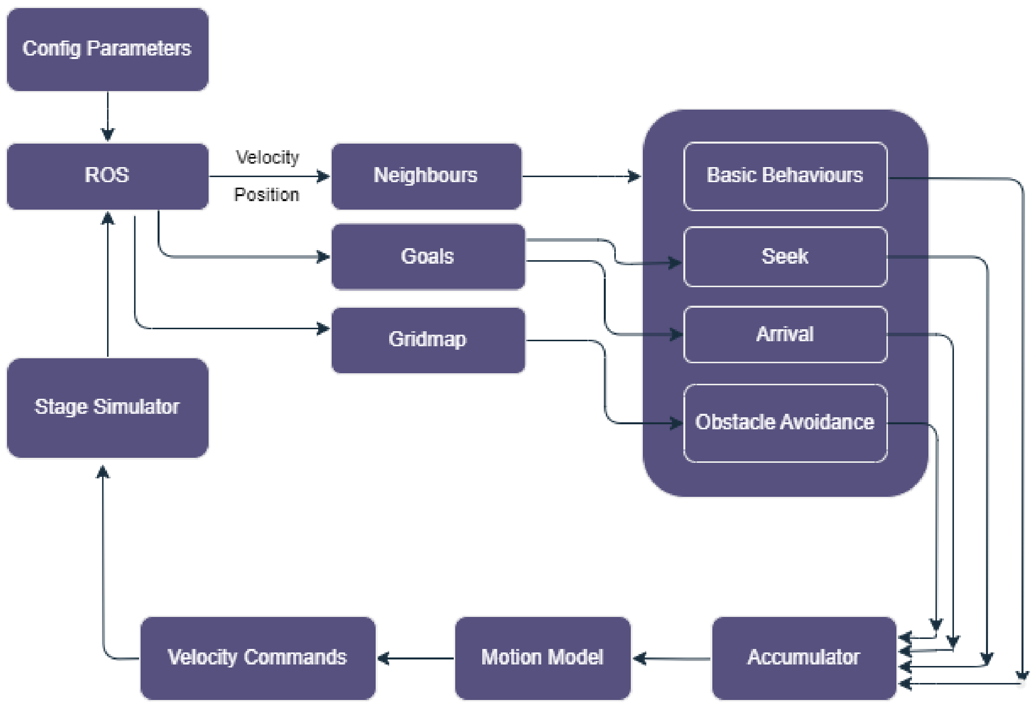
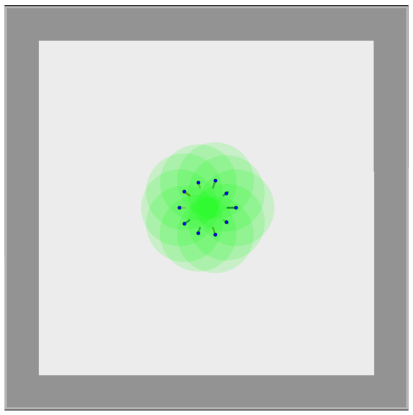
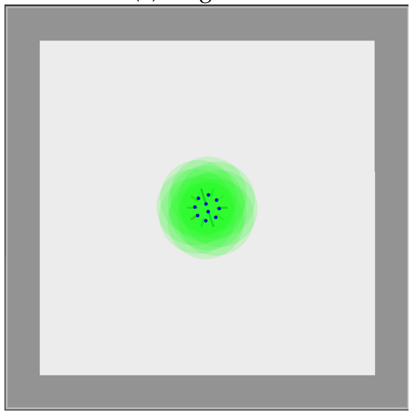


Fig. 13: Software Architecture

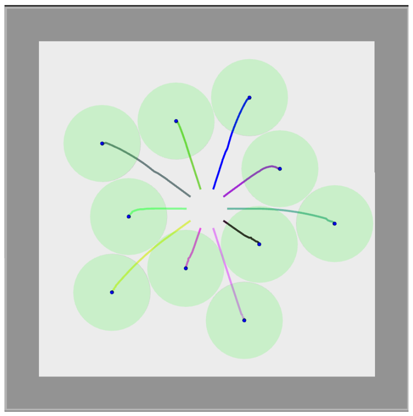


(a) Alignment



(b) Cohesion

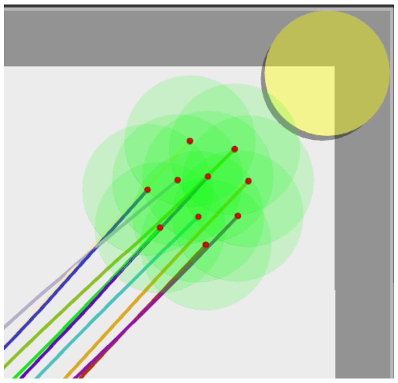
Fig. 14: Testing Basic Behaviors in an Empty Map.



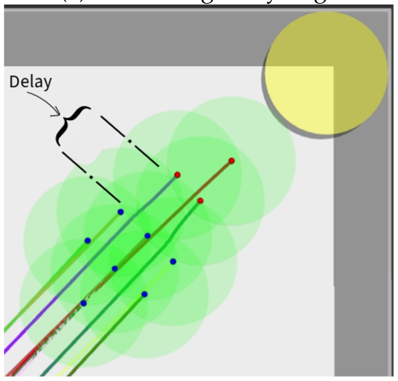
(c) Separation

Fig. 14: Testing Basic Behaviors in an Empty Map. (cont.) Boids are shown with blue dots and perceptive fields of view with transparent green circles.

Following the previous results, the arrival behavior was included in the accumulator to direct the boids to a set goal. In figure 15, we examine the effect of the number of leaders (i.e., global and local migratory urge) on the boids using the nearest neighbor method introduced in section 2.5. When the migratory urge affects all boids in the flock, the reaction of all the boids is instant. This can be deduced from the relatively equal distance between the boids as they proceed to the set goal. On the other hand, a flock of boids will act more naturally when the migratory urge is sensed by only a subset of the flock (i.e., a few "leaders"). We observe this in the gap between the leaders (red) and the rest of the flock (blue), which translates to a delay in the natural sense. The migratory urge also affects the convergence of the boids at the goal position.

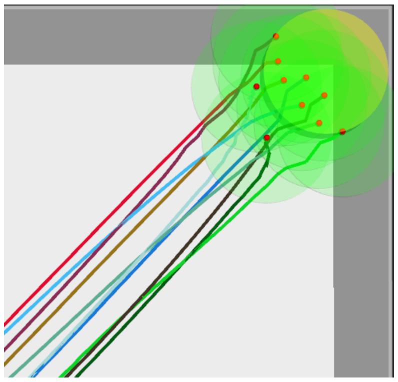


(a) Global Migratory Urge

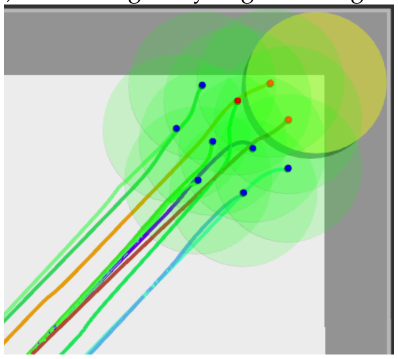


(b) Local Migratory Urge

Fig. 15: The Effect Varying the Number of Leaders on Flocking



(a) Global Migratory Urge Convergence



(b) Local Migratory Urge Convergence

Fig. 16: The Effect Varying the Number of Leaders on Arrival

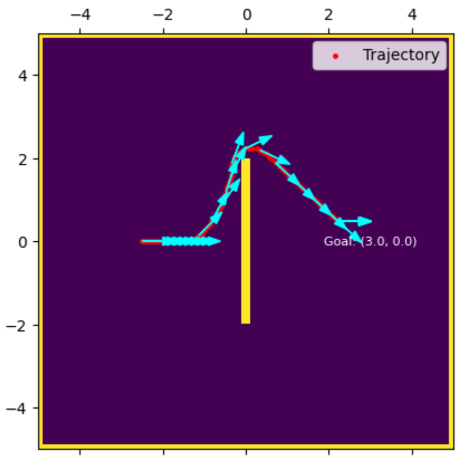
Under a local migratory urge, the leading boids will reach and stop at the goal while the followers will aggregate around the leaders. When all boids are equally influenced by the goal, they all compete to converge as close to the goal as possible. See figure 16.

4.2 Comparative Analysis

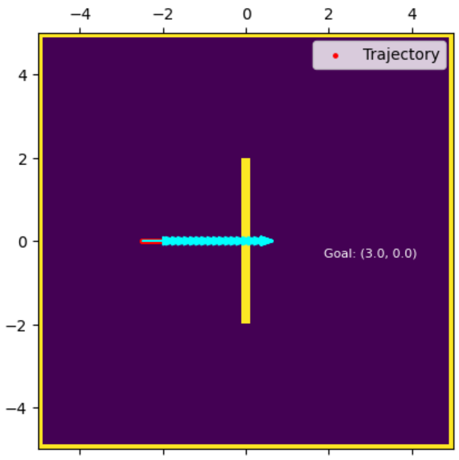
4.2.1 Potential field vs Steer-To-avoid

The two approaches considered for obstacle avoidance exhibit differing levels of effectiveness. Although the potential field method offers the advantage of low computational complexity, it is only because it is computed once assuming the map is known. For a real robot, the potential field is locally computed making it computationally expensive online. Adding to this, the potential field falls short in overall performance compared to the steer-to-avoid strategy. This discrepancy primarily arises from inherent corner cases that can emerge with a repulsive field.

Figures 17 and 18 illustrate the trajectory of a boid in a specific start-goal configuration within linear and circular environments, respectively. In both scenarios, steer-to-avoid adeptly guides the boid around the obstacle. However, the potential field method requires closer scrutiny.

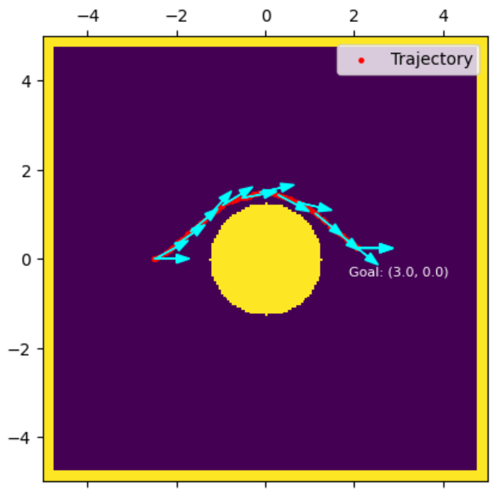


(a) Steer to Avoid

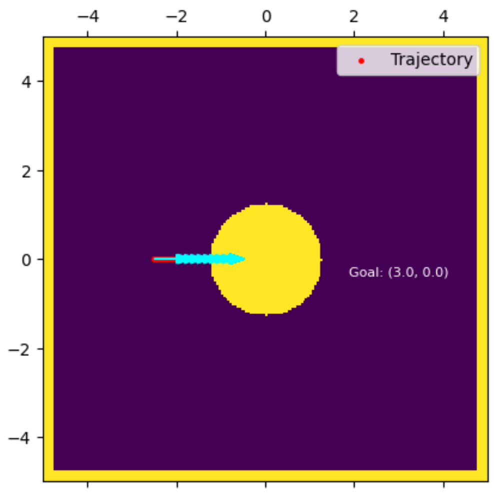


(b) Potential Field

Fig. 17: Obstacle Avoidance on Line Environment



(a) Steer-to-Avoid



(b) Potential Field

Fig. 18: Obstacle Avoidance on Circle Environment

In the potential field approach, when the boid enters the force field of the obstacle, the repulsive force from the obstacle acts directly opposite to the attractive force generated by the ”seek” behavior toward the goal. When these two vectors are combined, two possible outcomes emerge. If the attractive force outweighs the repulsive force in magnitude, the boid is repelled from the obstacle and exits its field, only to re-enter due to the attractive force. This results in oscillatory behavior near the boundary of the force field. In the alternative scenario where the repulsive force is smaller than the attractive force, the boid simply slows down and collides with the obstacle, as demonstrated in Figures 17b and 18b.

In Figure 19, an extreme corner case is depicted where the goal is positioned beyond the border of the map. The boid adeptly avoids collisions with steer-to-avoid, navigating alongside the wall without any collisions.

Given the inherent issues with the potential field approach and the superior characteristic of steer-to-avoid, which guides the boid along the obstacle rather than merely repelling it, the latter emerges as the undisputed choice for the obstacle avoidance algorithm in this work.

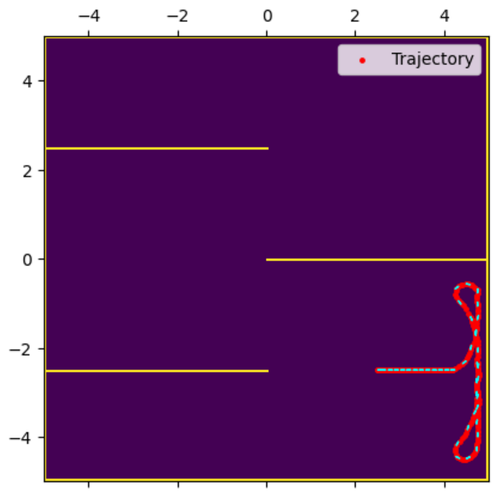


Fig. 19: Goal Beyond Wall - Steer-to-avoid

4.2.2 Weighted Acceleration vs. Prioritized Acceleration Accumulation

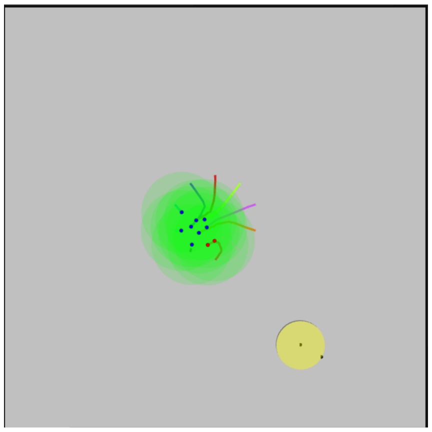
The weighted average method for acceleration accumulation produces a single steering force by taking the weighted sum of the acceleration requests from the behaviors under consideration. While this method is straightforward and easy to tweak, the equal treatment of all forces is not ideal. In this case, conflicting forces may cancel out leading to undesirable responses.

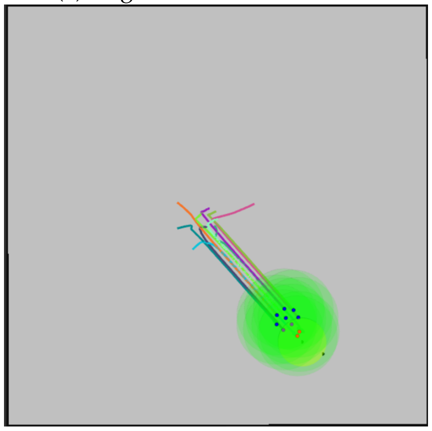
The prioritized acceleration accumulation (PAA) method, on the other hand, gives precedence to more important requests. This enables each boid to respond promptly to critical forces without being dampened by other forces. This approach is also highly scalable and better for comparative analysis. A major issue with this, however, is that prioritization is context- and environment-specific in most cases. Furthermore, PAA does not allow for *easy* priority reallocation which is required for highly complex environments.

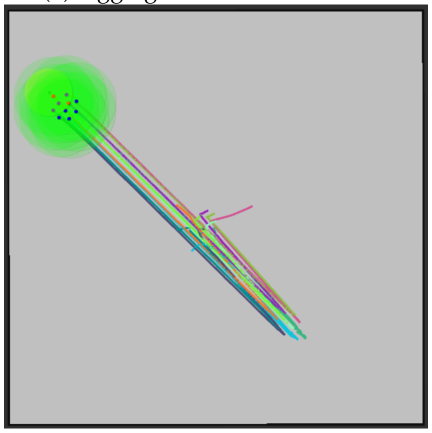
The performance of this method can be assessed in the accompanying videos of this report.

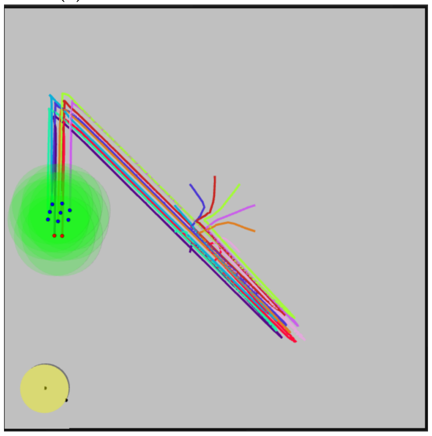
4.3 Multi-goal Aggregation

Building on the single-goal arrival behavior, we implemented a multi-goal arrival approach to aggregate the flock at several predetermined positions on the map. In our implementation, we assumed there was a list of goals that we wanted the flock to reach. Each goal acts as a new aggregation point, and the flock is directed sequentially from one goal to the next. The transition to a new goal occurs when a certain percentage of the flock reaches the current goal. This percentage is a predetermined parameter that can be tuned, thus ensuring smooth and continuous movement of the flock across different areas of the map. Figure 21 below shows the trajectory of each boid in the flock as they migrate to each designated aggregation point.


(a) Migration to the First Goal


(b) Aggregation at the First Goal


(c) Arrival at the Second Goal


(d) Migration to the Third Goal

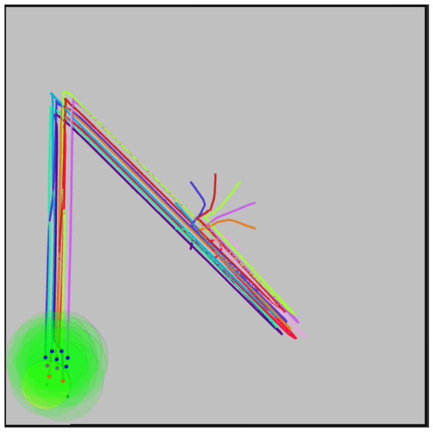

(e) Aggregation at the Third Goal

Fig. 21: Multi-goal Aggregation

4.4 Parameters Tuning

4.4.1 *Effect of Angle Incremental of Steer-to-avoid*

As previously mentioned, the steer-to-avoid behavior iteratively adjusts the angle to determine a new heading. This angle increment serves as a parameter and requires tuning.

In environments featuring lengthy wall obstacles aligned with the direction of motion toward the goal, a slightly higher increment angle is advantageous. This facilitates rapid turning without significantly expanding the obstacle's radius of influence. For instance, in a 'line' environment, a value of 25 degrees was employed.

Conversely, in cluttered environments, especially when dealing with small obstacles, a smaller increment angle is preferable. This allows for more fluid movements around the obstacles, akin to the agile maneuvers of birds. This scenario is notably observed in environments such as 'asteroid_1'.

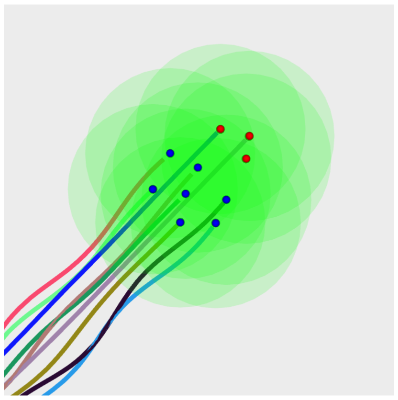
4.4.2 *Effect of Local Neigborhood's Field of View*

By adjusting the field of view (FoV) angle within the boid's local neighborhood to less than 360°, a bias is introduced towards boids in front, mimicking the limited vision of birds. This choice is intentional as birds predominantly look forward, avoiding influences from their rear.

Modifying the FoV significantly impacts the cohesive behavior of the flock. A 360° FoV promotes circular/elliptical movement, with forces experienced from all directions. In contrast, a reduced field of view results in a more streamlined flocking pattern, akin to the movements observed in birds. Figure 22 visually illustrates these behaviors.

4.4.3 *Parameter Tuning for Combined Behaviors in Different Map Environments*

Firstly, we consider a case where the focus is on keeping the boids flocking within a defined environment. Here, the migratory urge was removed in favor of the fundamental behaviors (alignment, cohesion, and separation) and obstacle avoidance (for the boundaries of the map). The hierarchy of behaviors for the prioritized acceleration allocator included obstacle avoidance (i.e., steer-to-avoid), separation, alignment, and cohesion, in that order. The first observation was that having obstacle avoidance as the prioritized task


(a) 360° FoV

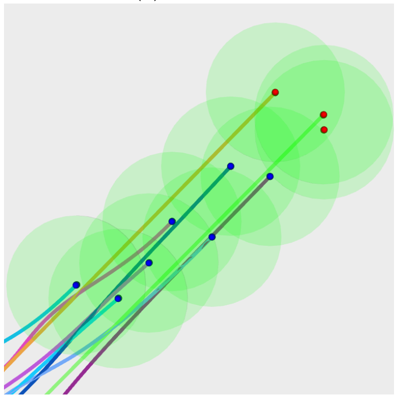

(b) 135° FoV

Fig. 22: Variation in FoV of Local Neighborhood

produced the smoothest flocking behavior. Additionally, this task received a high weight relative to the others. Separation was given the next priority to prevent impending collisions that stem from boids experiencing varying forces in response to perceived obstacles. Another key observation was that keeping the alignment weight lower relative to the other basic behaviors (with obstacle avoidance active) was imperative. This is underpinned by the fact that seamless obstacle avoidance in a swarm requires varying accelerations for different agents in the swarm. Therefore, velocity matching may lead to collisions if not appropriately weighted.

In the presence of multiple obstacles of varying shapes and sizes, the weight of alignment is set low or completely removed from the behavior hierarchy. This implementation choice results from the increased level of difficulty in the maneuvers required to successfully circumnavigate obstacles as a flock. This comment also holds true for large flocks in small environments. In a competition for space, while boids flock and avoid critical obstacles, alignment becomes counterintuitive.

Finally, since the arrival behavior generates an acceleration that constantly points to the goal, it tends to contradict other acceleration requests, especially those of obstacle avoidance. To mitigate this, we set the arrival weight as the lowest of all combined behaviors.

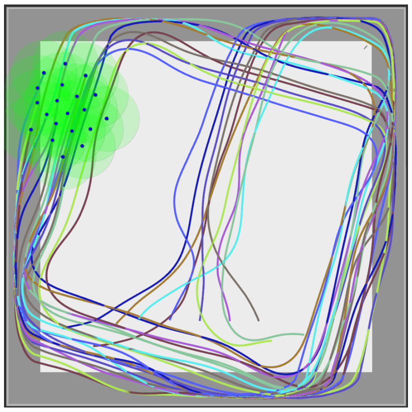
We now present some results from exhaustive tests on different environments. The parameters for each of these results are included in table 2.

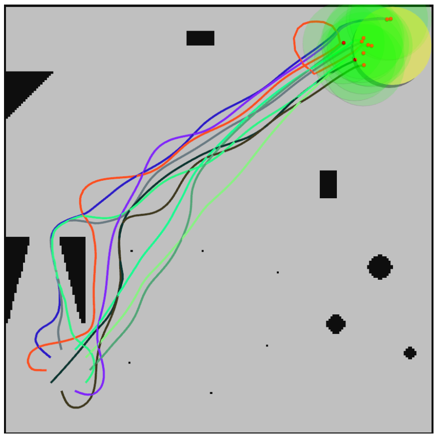
Fig.	w_{align}	w_{cohere}	$w_{separate}$	w_{steer}	w_{arrive}	Priority
23a	0.1	0.2	0.2	0.85	0.0	St>S>A>C
23b	0.0	0.35	0.2	0.85	0.1	Ar>St>S>A>C
23c	0.1	0.2	0.2	0.85	0.0	St>S>A>C
23d	0.1	0.35	0.2	0.85	0.1	Ar>St>S>A>C
23e	0.1	0.2	0.2	0.85	0.0	St>S>A>C
23f	0.0	0.25	0.2	0.85	0.1	Ar>St>S>A>C

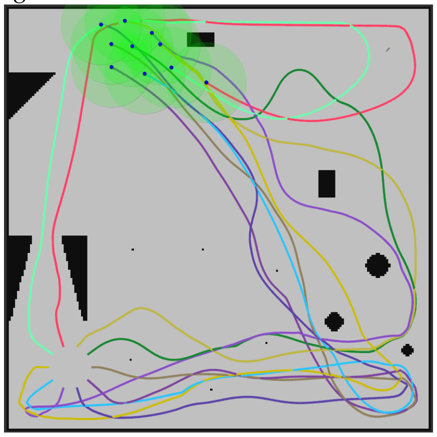
TABLE 2: Simulation with Combined Behaviors (for fig. 23). Key: St - Steer-to-avoid; S - Separation; Ar - Arrival; A - Alignment; C - Cohesion.

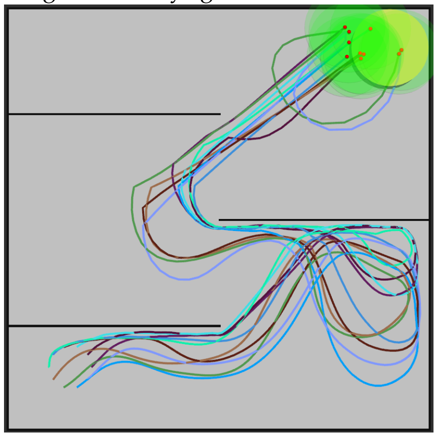
5 Conclusion and Future Work

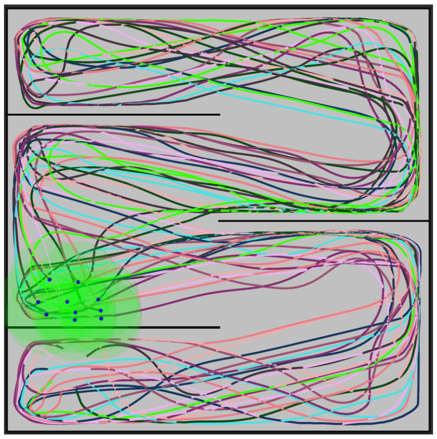
To conclude, this project aimed to implement Reynolds' rules for swarm control in a simulated environment using robots. Initially, the basic Reynolds' rules—separation, alignment, and cohesion—were successfully implemented and tested in the Stage simulator within a mapped frame. In this phase, we expanded the Reynolds' rules to include navigation to specific points and obstacle avoidance for the swarm. These additions aimed to enhance the swarm's capabilities by enabling it to navigate towards designated points while gradually reducing speed as it neared the goal, and to actively evade obstacles as they came into proximity. The thorough testing across diverse maps, encompassing various obstacles and maze structures, allowed us to evaluate how well these rules performed and adapted within different scenarios. Moving forward, the transition to hardware implementation stands as the future work for this project. Real-world application of these refined rules and innovative algorithms on physical robots would validate their practicality and efficiency, opening doors for their integration into various industries where swarm intelligence could bring about substantial advancements.


(a) 20 Boids Flocking with Obstacle Avoidance. As the number of boids increases, more precisely tuned parameters are required to ensure fluid movements, especially at corners where the allowed turned radius may be too small for a large flock.


(b) 10 Boids Migrate to a Set Goal while Avoiding Obstacles. Given that the obstacles are not long stretches of walls (like in mazes), arrival at the set goal is always guaranteed on this map.


(c) 10 Boids Flocking with Obstacle Avoidance. This shows the boids successfully avoid obstacles of different shapes and sizes, bifurcating and converging where necessary.


(d) 10 Boids Migrate to a Set Goal while Avoiding Obstacles. The effect of the arrival vector is evident by the trajectory of the boids (predominantly oriented toward the goal).


(e) 10 Boids Flocking with Obstacle Avoidance. At the highlighted corner, the boids approach from two different directions due to the bifurcation and subsequent convergence that occurred.

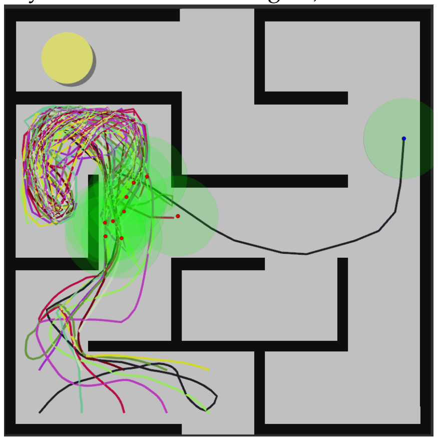
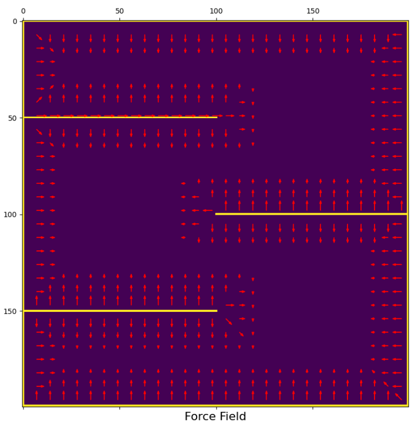

(f) 10 Boids Migrate to a Set Goal while Avoiding Obstacles. In this hard maze map, the boids mainly flock within a specific area of the map due to the strong forces of the surrounding walls and the goal (yellow) on the other side. This shows that the boid algorithm is primarily not designed for maze-like environments.

Fig. 23: Simulation Results in Different Maps

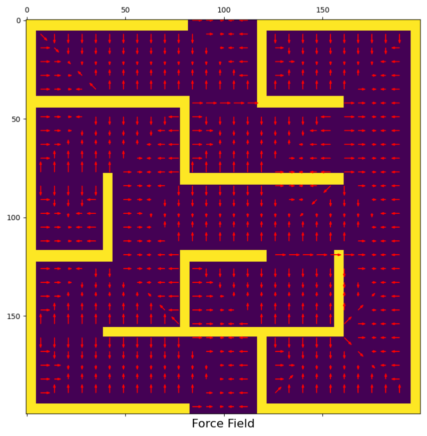
References

[1] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware," SIGGRAPH '90, (New York, NY, USA), p. 327–335, Association for Computing Machinery, 1990.

Appendix A
Potential Field



(a) Simple Maze Environment



(b) Hard Maze Environment

Fig. 24: Maze Environments