Controlling a Swarm of Robots in 2D Simulator Stage using Reynolds' Rules

Fatima Yousif Rustamani, Pham Than Loc, Sawera Yaseen, Tanakrit Lertcomepeesin, and Vania Katherine Mulia

Abstract—This project aims to implement Craig Reynold's behavioral rules to control a robot swarm within a 2D simulator Stage. The implementation begins with integrating basic Reynold's rules such as Separation, Alignment, and Cohesion, followed by additional rules such as Navigation to a certain point and Obstacle avoidance. The robots simulated in the Stage simulator are omnidirectional points, that are evaluated in various maps to analyze their behavior across different environments. The simulation results of the project can be found here.

Index Terms—Robot Swarm, Flocking, Reynolds' Rules, Separation, Alignment, Cohesion, Obstacle Avoidance

1 Introduction

Robot swarms have become increasingly popular in the field of multi-robot systems. A robot swarm is characterized by multiple robots, usually with rather simple capabilities, moving together in coordination. To achieve coordination, each robot relies on local interactions with other robots and with the environment within its perceptive field. Generally, swarm intelligence draws inspiration from nature, such as flocks of birds, swarms of ants, and schools of fish.

Craig W. Reynolds in his 1987 work introduced a set of behaviors to simulate the behavior of a single entity, modeled as a boid, moving in a swarm [1]. This set of behaviors is known now as the Reynolds' Rules. The three basic behaviors are separation (collision avoidance), alignment (velocity matching), and cohesion (flock centering). These behaviors allow agents to form a group and move together. Additional behaviors, such as seek and arrival, allow the swarm to navigate to a certain point. Furthermore, an obstacle avoidance behavior allows each agent to avoid collision with obstacles in its environment.

This project aims to implement the Reynolds' Rules to simulate a robot swarm. In this project, a single robot is modeled as an omnidirectional entity. The algorithm is simulated in a 2D simulator Stage within the Robot Operation System (ROS) framework. The simulator includes different environment configurations to test out different behaviors. The simulation results provide insight into the dynamic behaviors of robots in a swarm.

Paper submitted on 06/Dec/2024

2 Methodology

2.1 Agent Model Definition

In this simulation, the model of a single agent (or boid) is encapsulated in a python class corresponding to a single ROS node. The python class *Boid* contains the attributes related to its kinematic properties, including its position, heading, velocity, maximum velocity, and maximum acceleration. The *Boid* class also contains the definition of its perception range and its neighboring boids (i.e. other boids within its perception range). The perception of a boid will be explained further in the next section.

In terms of its kinematics, the boid is modeled as a two-dimensional omnidirectional agent. This means that a single agent can move along the x and y axes independently. The kinematics of a single agent can therefore be defined as a simple two-dimensional kinematics model:

$$x_i = x_{i-1} + v_{x,i}\Delta t \tag{1}$$

$$y_i = y_{i-1} + v_{y,i}\Delta t \tag{2}$$

where x and y correspond to the position of the boid, with the subscripts indicating the time step, $v_{x,i}$ and $v_{y,i}$ are the velocity in the x and y directions, respectively, at the current time step, and Δt is the sampling time.

The kinematics model as described above does not include the boid's heading. However, the heading is needed to define the perception range. For this

purpose, the heading is defined based on the boid's velocity vector, expressed mathematically as

$$\theta_i = atan2(v_{y,i}, v_{x,i}) \tag{3}$$

2.2 Local Neighborhood

The boid's perception model, also referred to as its local neighborhood, is determined by a set distance, defining when two boids are considered "nearby," and an angle, which establishes the boid's perceptual "field of view".

The distance is defined as the Euclidean distance between two boids. For two boids with positions (x_i, y_i) and (x_j, y_j), the distance is expressed mathematically as

$$d = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{4}$$

. The other boid will be considered a neighbor only if this distance is lower than the boid's perception range.

The angular disparity between two boids with positions (x_i, y_i) and (x_j, y_j) can be calculated using the *atan2* function. Furthermore, the angular disparity is defined relative to the boid's heading. This computation is mathematically expressed as

$$\alpha_{ij} = atan2(y_j - y_i, x_j - x_i) \tag{5}$$

$$\beta_{ij} = |\alpha_{ij} - \theta_i| \tag{6}$$

where β_{ij} indicates the angle of the boid with subscript j with respect to the boid with subscript i, and θ_i is the current heading of boid i. With this expression, boid j will be considered a neighbor of boid i only if β_{ij} is lower than the field-of-view angle.

In the simulation, each boid initially receives information regarding the position of all other boids (via ROS topic subscriptions). The perception model is simulated in a class method that iterates through all other boids to compute the distance and angular disparity of the boid. The other boid is considered a neighbor only if the distance and angular disparity are within the perception range and angle. The position and velocity of all neighbors are then stored in a list, which will be used to control the boid's motion based on the behaviors in the Reynolds' Rule. This pipeline can be seen in Algorithm 1.

2.3 Reynolds' Basic Behaviors

The three core behaviors - separation, cohesion, and alignment - relate to how groups of boids interact. Each behavior specifies how a boid responds to others within its local neighbourhood, while boids outside this range are disregarded.

Algorithm 1 Local Neighborhood Algorithm

```
neighbors_list ← []
for each boid_j in the map do
    distance ← sqrt((x_i − x_j)^2 + (y_i − y_j)^2)
    β ← abs(atan2(y_i − y_j, x_i − x_j) − θ_i)
    if distance < perception_range and β < perception_angle then
        neighbors_list.append(boid_j)
    end if
end for
return neighbors_list
```

2.3.1 Separation

Separation behavior gives a boid the ability to maintain a certain separation distance from neighbouring boids, preventing boids from crowding together. The separation behavior can be mathematically represented as:

$$\text{Separation} = \frac{1}{N} \sum_{j=1}^{N} \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|^2}$$

where N denotes the number of neighboring boids, and the denominator denotes the Euclidean distance between the current boid $\mathbf{x}_i$ and its neighboring boids $\mathbf{x}_j$. The implementation of this behavior can be seen in Algorithm 2.

Algorithm 2 Separation Algorithm

```
Initialize sep_acc ← Point(0, 0)
for each neighboring boid do
    dis ← distance(self, neighbouring_boid)
    if dis > 0 then
        sep_acc ← sep_acc + (self.position − neighbouring_boid.position) / dis^2
    end if
end for
if magnitude(sep_acc) > 0 then
    sep_acc ← normalize(sep_acc)
end if
```

2.3.2 Alignment

Alignment behaviour allows aligning the heading direction of a boid with the average heading direction of its neighbors. The alignment behavior can be expressed as:

$$\text{Alignment} = \frac{1}{N} \sum_{j=1}^{N} (\dot{x}_j - \dot{x}_i)$$

where N denotes the number of neighboring boids, $\dot{x}_j$ represents the velocity (change in position) of the

j-th neighboring boid. The implementation of this behavior can be seen in Algorithm 3.

Algorithm 3 Alignment Algorithm

```
Initialize xvel_avg ← 0, yvel_avg ← 0, neighboring_boids ← 0
for each neighboring boid do
    if distance to the boid < visible_range then
        xvel_avg += otherboid.vx
        yvel_avg += otherboid.vy
        neighboring_boids += 1
    end if
end for
if neighboring_boids > 0 then
    xvel_avg ← xvel_avg/neighboring_boids
    yvel_avg ← yvel_avg/neighboring_boids
end if
Update the velocity:
boid.acc_x ← (xvel_avg − boid.vx)
boid.acc_y ← (yvel_avg − boid.vy)
```

2.3.3 Cohesion

Cohesion or flock centering behaviour allows boids to move towards the average position (or center of gravity) of neighboring boids within a defined spatial range. Mathematically, cohesion can be expressed as:

$$\text{Cohesion} = \frac{1}{N}\sum_{j=1}^{N}(\mathbf{x}_j - \mathbf{x}_i)$$

where N denotes the number of neighboring boids, $\mathbf{x}_i$ signifies the position of the current boid, and $\mathbf{x}_j$ represents the position of the j-th neighboring boid. The implementation of this behavior can be seen in Algorithm 4.

Algorithm 4 Cohesion Algorithm

```
Initialize xpos_avg ← 0, ypos_avg ← 0, neighboring_boids ← 0
for each neighboring boid do
    if distance to the boid < visible_range then
        xpos_avg += otherboid.x
        ypos_avg += otherboid.y
        neighboring_boids += 1
    end if
end for
if neighboring_boids > 0 then
    xpos_avg ← xpos_avg/neighboring_boids
    ypos_avg ← ypos_avg/neighboring_boids
end if
Update the velocity:
boid.acc_x ← (xpos_avg − boid.x)
boid.acc_y ← (ypos_avg − boid.y)
```

2.4 Navigation

The three basic behaviors in the Reynolds' Rules provide the boids with the capability to form a group. In order to move the flock to a target position, we implement a navigation behavior. This behavior can be described as a combination of two simpler behaviors: seek and arrival (as described in [2]).

2.4.1 Seek

The seek behavior is a fundamental steering mechanism in which an entity adjusts its motion to move toward a specific target. In the implementation of seek behavior the objective is to direct the boid towards its target position by calculating steering acceleration. The approach begins by determining the desired velocity, which is proportional to the vector difference between the goal position and the boids's current position. This proportionality is scaled using a gain factor, *nav_gain*, to control the responsiveness of the boids's motion. The desired acceleration is then computed as the difference between the desired velocity and the boid's current velocity, ensuring that the boid transitions smoothly toward the target. Furthermore, to maintain a realistic flocking motion the acceleration is constrained using the limiting function, preventing abrupt changes, resulting in efficient and stable navigation of the boid. The implementation of this behavior can be seen in Algorithm 5.

Algorithm 5 Seek Behavior: Navigation Acceleration

```
1: nav_acc ← Point(0, 0)
2: nav_vel ← Point(0, 0)
3: if goal exists then
4:     nav_vel ← (goal − position) × nav_gain
5:     nav_acc ← nav_vel − velocity
6: end if
7: return LIMIT(nav_acc)
```

2.4.2 Arrival

The arrival behavior resembles the seek behavior in a way that both behaviors guide the flock towards the target from afar. However, while seeking directs the boid to move towards the target, arrival focuses on ensuring a smooth and gradual deceleration as the boid approaches the goal, ultimately stopping it at the target position. The arrival method is implemented to achieve this by first calculating the offset vector from the boid's current and goal positions and determining the Euclidean distance to that goal position. When this distance falls below a specified threshold, it indicates that the goal has been reached, resulting in no acceleration. Additionally, to enable

the boid to decelerate as it approaches the goal, the method calculates the ramped speed proportional to the distance. This speed is clipped to ensure that it does not exceed the maximum velocity of the boid self.max_vel. The velocity vector is then normalized according to the distance, directing the boid towards the goal position with a controlled rate. The difference between the desired velocity and current velocity is then calculated to find the required acceleration. This acceleration is also limited to ensure it stays within safe and realistic limits producing a smooth arrival behavior. The implementation of this behavior can be seen in Algorithm 6.

Algorithm 6 Arrival Steering Behavior

```
target_offset ← goal_position − current_position
distance ← LENGTH(target_offset)
if distance < tolerance then
    return zero acceleration
end if
ramped_speed ← max_velocity × (distance / tolerance)
clipped_speed ← MIN(ramped_speed, max_velocity)
desired_velocity ← (clipped_speed / distance) × target_offset
arrival_acceleration ← desired_velocity − current_velocity
return LIMIT(arrival_acceleration)
```

2.5 Obstacle Avoidance

The fundamental behaviors described above ensure cohesive flock movement among the boids while preventing collisions with each other. However, the environment may introduce obstacles that make it necessary for the boids to use avoidance maneuvers. In the following subsection, steer to avoid strategy is presented to address this crucial aspect of behavioral response.

2.5.1 Steer to Avoid

The steer-to-avoid method helps avoid obstacles by checking possible paths from a "lookahead point". This point is a predicted future position of the object (boid), based on its current direction and any changes it might make to its angle.

The algorithm adjusts the orientation angle in small steps, exploring both rightward and leftward directions from the current orientation. For each direction, it calculates a new angle and generates a lookahead path starting from the object's current position.

These calculated paths are then prepared for checking validity. The algorithm evaluates whether these paths encounter any obstacles 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 until a valid, obstacle-free path is found or until the angular adjustments reach a limit without finding a clear path. This terminates the algorithm, indicating the unavailability of a valid path that avoids obstacles.

In the algorithm shown in Algorithm 7, we check the whole path ahead instead of only the look-ahead point. Because path checking occurs in stepwise increments along the line, it also addresses the unavailability of the look-ahead point.

Algorithm 7 Steer-to-avoid Algorithm

```
while not path valid do
    α ← α + step angle
    newθ ← θ_v + α
    Calculate the lookahead path based on newθ
    Check if the new path is valid
    if path is valid then
        Update the orientation & exit loop
    else
        newθ ← θ_v − α
        Calculate the lookahead path based on newθ
        Check if the new path is valid
        if path is valid then
            Update the orientation & exit
        end if
    end if
end while
return θ_boid
```

Besides the path validity checking follows the algorithm in Algorithm 8

2.6 Combining Behaviors

All behaviors explained in the previous sections produce a required acceleration to fulfill each behavior. To combine the acceleration from all behaviors, we implemented the priority acceleration allocation algorithm as described in [1]. In this approach, all behaviors are ranked in a priority list. The algorithm loops through the priority list to accumulate the acceleration from all behaviors, from higher to lower-priority behavior, until the magnitude of the acceleration reaches the maximum limit set for each boid.

While implementing the algorithm, we also noticed that the acceleration from all behaviors are not in the same scale. This would cause problems, as one

Algorithm 8 Check Path Function

```
dist ← CalculateDistance(p1, p2)
num_steps ← dist/step_size
num_steps ← integer value(num_steps)
for j ← 0 to 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))
end for
waypoints ← waypoints[1 :]
for w in waypoints do
    if IsValid(w) is False then
        return False
    end if
end for
return True
```

behavior may "overshadow" other behaviors due to relatively large acceleration differences. To overcome this, we have slightly modified the algorithm to include weights for each behavior.

The implemented algorithm can be seen in Algorithm 9.

Algorithm 9 Priority Acceleration Allocation Algorithm

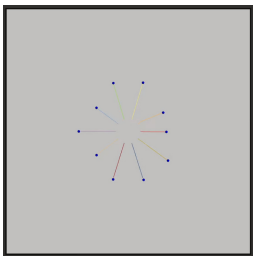
```
1: Input: acceleration from all behaviors
2: priority_list ← list of accelerations from all behavior
3: Define weight_list
4: acceleration ← 0
5: for i in range(len(priority_list)) do
6:     acceleration += weight_list[i] × priority_list[i]
7:     if norm(acceleration) > max_acceleration then
8:         break
9:     end if
10: end for
11: return acceleration.clip(max_acceleration)
```

3 Results and Discussion

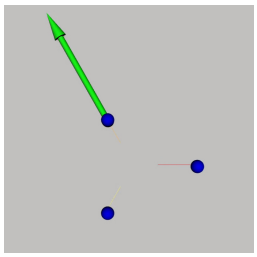
3.1 Simulated Behaviors

3.1.1 Reynold's Basic Behaviors

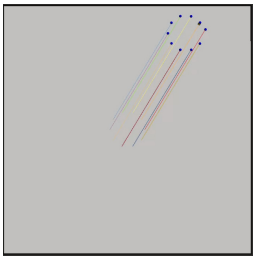
The Figure 1 shows testing done in an empty map to see the individual affects of the three basic behavior results from simulation with plotted trajectories and their respective accelerations shown with green, yellow and blue arrows for each behavior (separation, alignment and cohesion). The boids are shown with blue dots. In these results, we observe that while separation and cohesion act in opposite directions, alignment is only ensuring that boid is moving at the same velocity as its neighbors which is also shown in the equal trails of displacement in Figure 1c. When acting almost in isolation, separation moves boid clear of its neighbors. With accelerations of these three behaviors along with navigation and obstacle avoidance combined acceleration is produced which helps boids eventually converge to a formation.



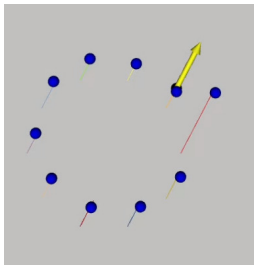
(a) Separation in Simulation



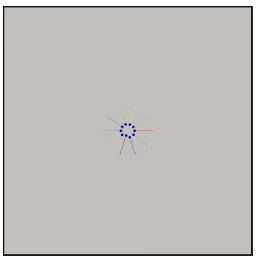
(b) Separation acceleration



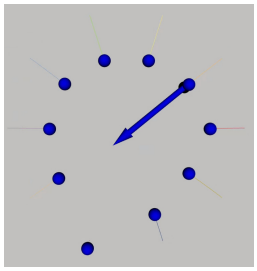
(c) Alignment in simulation



(d) Alignment acceleration



(e) Cohesion in simulation

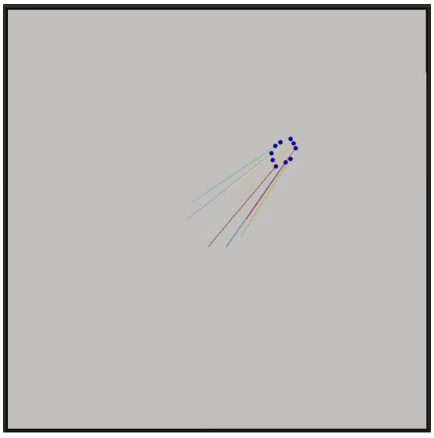


(f) Cohesion acceleration

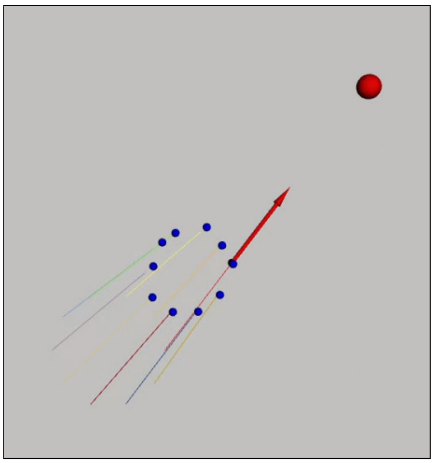
Fig. 1: Basic behaviors in simulation with plotted trajectories and their accelerations marker (arrows)

3.1.2 Navigation

The effect of the two navigation behaviors i.e. seek and arrival on the motion of boids are in affect that the



(a) Seek simulation



(b) Seek acceleration pointed with arrow alongside the goal represented with red marker

Fig. 2: Navigation behaviors including seek and arrival

seek's objective is to propel the flock towards a target at maximum speed and later's objective is to slow down the flock when approaching the goal. While the seek behavior results are shown with simulation results in Figure 2a and acceleration pointed with an arrow in figure 2b. The effect of arrival behavior can be best visualized in the video here, where the flock leader is notably slowed down when approaching the goal.

3.1.3 *Obstacle avoidance*

The implemented obstacle avoidance behavior gives each boid the ability to steer away from the obstacles without getting trapped facing the obstacle head-on. The simulation of this behavior can be seen in Figure 3. With the steer-to-avoid approach, instead of generating a steering force in a completely opposite direction to the relative position of the obstacle, the generated acceleration allows the boid to evade the obstacle ahead of it smoothly, as seen in figure 3a. This results in a more natural motion of the flock, as seen in figures 3b and 3c.

3.2 Effect of Parameters

3.2.1 *Effect of Perception Range*

To observe the effect of perception range (i.e. the radius of perception) on the motion of the boids, the algorithm was tested using three different values of perception range: 0.6, 1.2, and 2.4. The simulation results can be seen in Figure 4.

Generally, having greater perception range means each boid will be able to "observe" more neighbors, providing the boid with more references to adjust its acceleration to form a flock. It can be observed that for a perception range of 0.6 (figure 4a), the boids fail to form a single flock. It is interesting to notice, that the boids still exhibit a tendency to form groups, as indicated by several boids that are very close to each other moving together. This is the result of all the behaviors in the Reynolds' Rules. However, the perception range is so small that some boids observe too few or even no neighbors at some point, resulting in deviations from the flock.

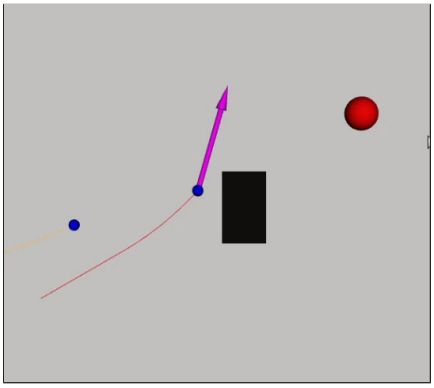
A perception range of 1.2 (figure 4b) proves to be sufficient for most of the boids to form a single flock, with only a couple boids deviating from the flock. For a perception range of 2.4 (figure 4c), all the boids successfully move together in a single flock.

3.2.2 *Effect of Field-of-View Angle*

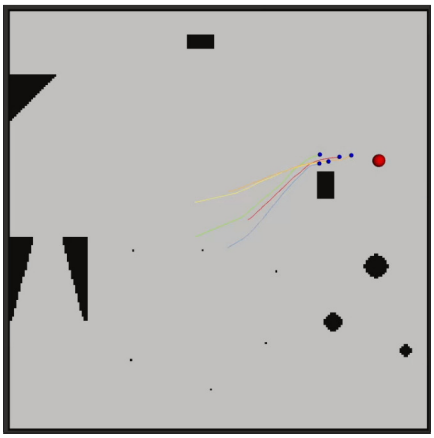
To observe the effect of field-of-view (FoV) angle (i.e. the angle of perception) on the motion of the boids, the algorithm was tested using three different values of FoV angle: $\frac{\pi}{2.2}$, $\frac{\pi}{1.4}$, and π. The simulation results can be seen in Figure 5.

There are two observed effects of the FoV angle. The first observed effect occurs when the FoV angle is set to $\frac{\pi}{2.2}$ (figure 5a). In this scenario, the FoV angle is so small that a boid can lose track of other boids, especially during turning motions. As a result, some boids separate from the group.

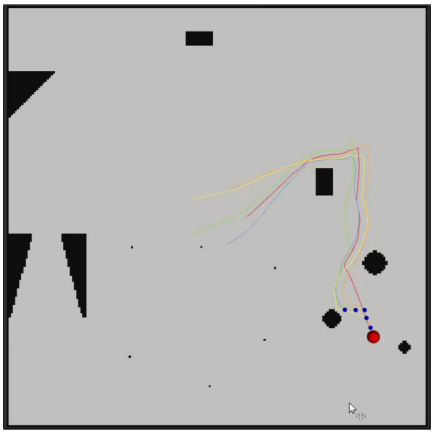
The second observed effect can be seen when comparing the formations generated with FoV angles of $\frac{\pi}{1.4}$ (figure 5b) and π (figure 5c). Since the FoV angle is measured from the boid's heading, having a FoV angle of π means having a $360\,\mathrm{deg}$ view. This implies that a boid can see other boids in its back (i.e. not forward-focused). This results in the boids moving in a circular formation, which is unnatural. With a FoV angle of $\frac{\pi}{1.4}$, the boid's perception is more forward-focused,



(a) Acceleration from obstacle avoidance behavior (arrow marker)

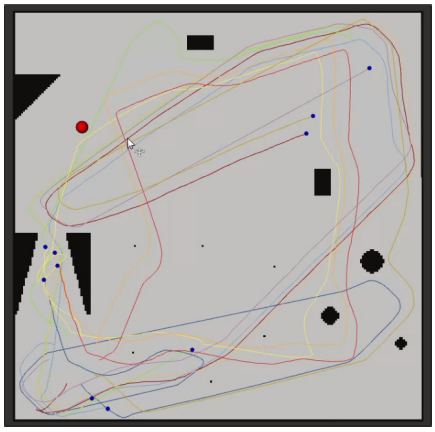


(b) Simulation of obstacle avoidance behavior (obstacle 1)

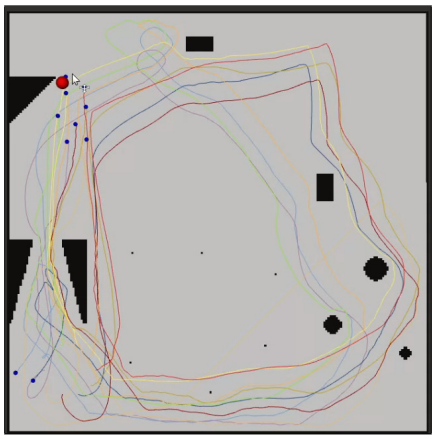


(c) Simulation of obstacle avoidance behavior (obstacle 2)

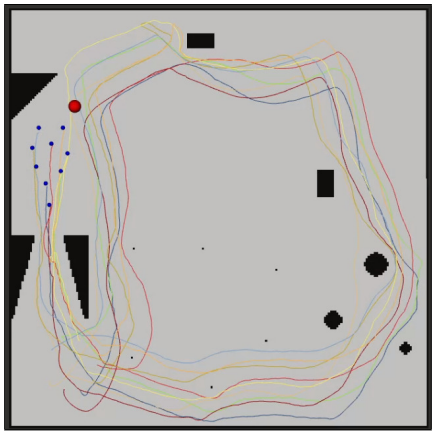
Fig. 3: Simulated Obstacle Avoidance Behavior



(a) Perception range = 0.6



(b) Perception range = 1.2



(c) Perception range = 2.4

Fig. 4: Effect of Different Perception Range on Boids' Trajectories

resulting in a more natural formation in which each boid follows the boids in front of it.

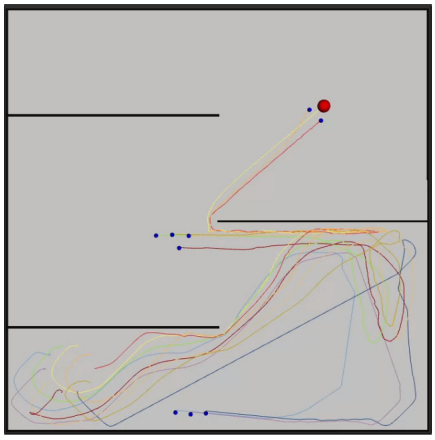
3.2.3 Effect of Different Behavior Prioritization

Scenario	Figure	Priority Order
1	Figure 6	Ob_Av > S > Ar > Se > A > C
2	Figure 7	C > A > Ar > S > Ob_Av > Se

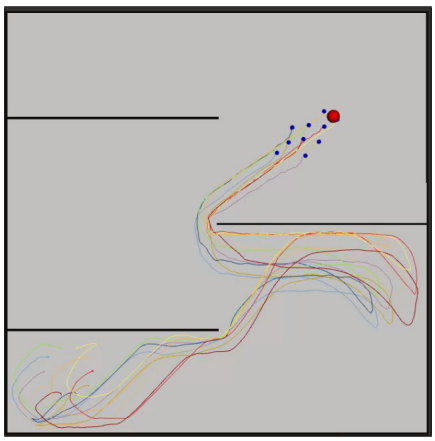
TABLE 1: Priority allocations for different scenarios Key: A = Alignment, Ar = Arrival, S = Seek, Ob_Av = Obstacle Avoidance, C = Cohesion, Se = Separation

The Figure 6 shows scenario 1 of prioritized acceleration allocation mentioned in Table 1. In this scenario, boids navigate through a maze-like environment toward the goal position. This scenario focuses on the role of safety-focused behaviors. This prioritization ensures that boids effectively navigate around obstacles (walls) and maintain a safe distance from each other, by temporarily sacrificing the lower-priority behaviors alignment and cohesion. As a result, it can be seen from the boids' trajectories that the boids diverge in regions of the maze where walls are present. This divergence allows the boids to adaptively map different paths while maintaining collision-free trajectories, as observed in areas with sharp turns. It can also be noted that boids do not converge as they approach the goal. This lack of convergence highlights the trade-off between safety-focused behaviors (Obstacle avoidance and separation) and group cohesion. While the system ensures the boids progress towards the target point, the prioritization of obstacle avoidance causes a loss of cohesion and alignment in this scenario. This can be seen near narrow passages where each boid takes an independent trajectory toward the goal with very little influence from low-priority behaviors. This scenario shows the robustness of prioritized allocation where safety is critical. Although basic flocking behaviors like alignment and cohesion are sacrificed in this scenario, the system ensures that all boids reach the goal safely, adapting to environmental constraints.

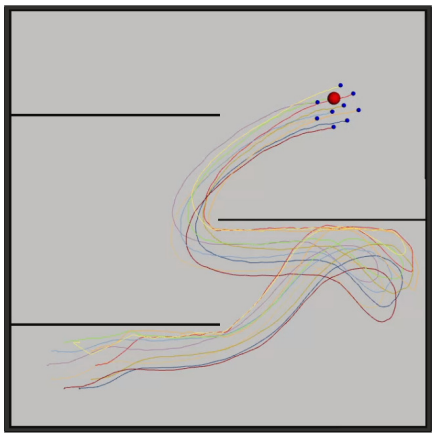
The Figure 7 shows scenario 2 of prioritized acceleration allocation mentioned in Table 1. Contrary to previous scenario, where safety behaviors dominated, this scenario focuses on maintaining the group dynamics of the boids. As a result, Agents stay closer to each other as they navigate towards the goal position. The emphasis on the higher priority of alignment and cohesion is evident in narrow paths as boids prioritize moving together as a group, making the flock's motion synchronized and consistent. However, putting obstacle avoidance and separation behaviors on lower priority introduces safety trade-offs. The trajectories show some instances where boids closely



(a) FoV angle = $\frac{\pi}{2.2}$



(b) FoV angle = $\frac{\pi}{1.4}$



(c) FoV angle = π

Fig. 5: Effect of Different Field-of-View Angles on Boids' Trajectories

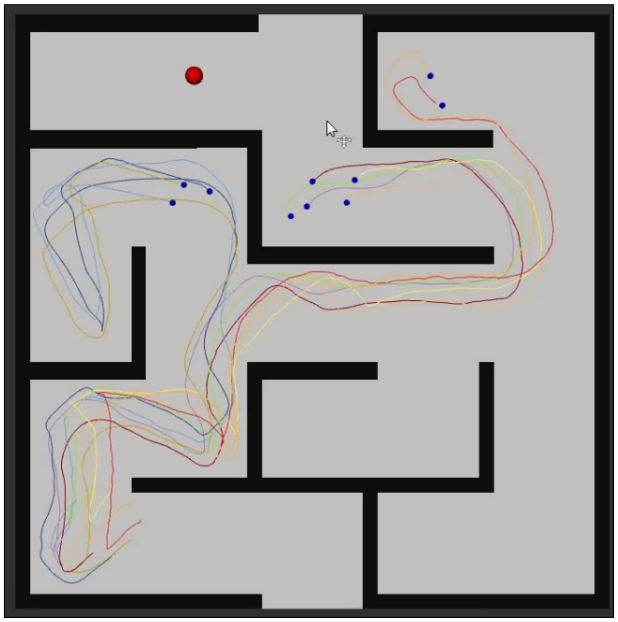


Fig. 6: Scenario 1 of Behavior Prioritization

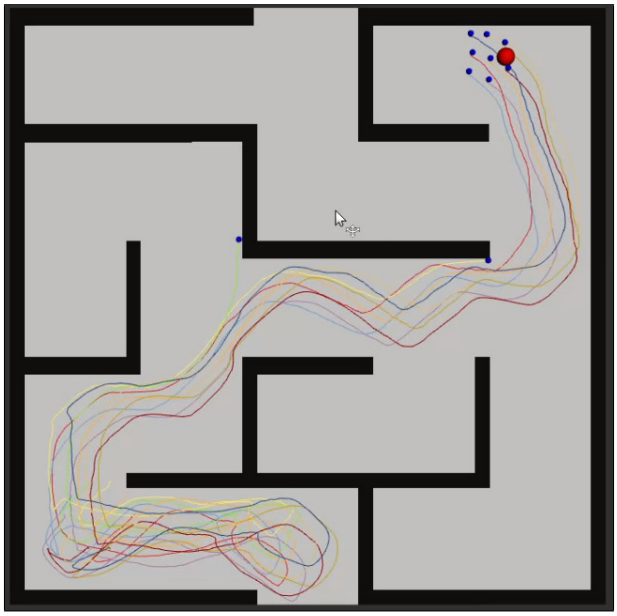


Fig. 7: Scenario 2 of Behavior Prioritization

approach the walls of the maze or move in proximity to each other. Despite these trade-offs, the scenarios depict goal-seeking behavior, as the boids collectively navigate toward the goal and converge to it. The convergence highlights the influence of cohesion and alignment, coupled with arrival and seek behaviors, in driving the group toward their objective. To conclude, this scenario demonstrates the impact of prioritizing cohesion and alignment over obstacle avoidance and separation. The resulting behavior of the boids is more group-oriented, they maintain a cohesive structure while moving toward the goal. However, this comes at the cost of reduced emphasis on safety behaviors, which could lead to increased collision risks in environments with tighter constraints or moving obstacles.

4 Conclusion

In this phase of the project, we have successfully implemented the three basic behaviors in the Reynolds' Rules, namely separation, alignment, and cohesion. The combination of these behaviors guides the agents to form a swarm. Moreover, the addition of navigation behavior, which is a combination of seek and arrival behaviors, guides the swarm to move together to a desired position. Furthermore, the obstacle avoidance behavior enables the agents to evade obstacles ahead of them. The steer-to-avoid approach proves to be a more favorable approach to obstacle avoidance because it allows for a more natural motion of avoiding collision with obstacles.

This project also explores the effect of several parameters, namely the perception range, field-of-view (FoV) angle, and behavior prioritization. An increase in perception range leads to a higher ability of all boids to gather and move together in a single flock. While an increase in FoV angle generally leads to a higher capability of maintaining the flock while moving, FoV angle too high causes unnatural formation of the flock. Finally, setting an appropriate behavior prioritization and proper tuning of the weights of each behavior is important, as different behavior prioritization results in different behaviors of agents within the flock.

Future work for this project includes hardware implementation of this algorithm into robotic platforms.

References

[1] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, Aug. 1987, doi: https://doi.org/10.1145/37402.37406.

[2] C. W. Reynolds, "Steering Behaviors For Autonomous Characters," *www.red3d.com*, 1999. https://www.red3d.com/cwr/steer/gdc99/ (accessed Dec. 05, 2024).