Abstract
This report compares two steering behavior algorithms in crowded path following. Both steering and path following could be a part of a flocking simulation. The report is divided into two parts, one introductory part which explains what a flocking simulation is, what the goal of the report is and what methodology is used to acquire that goal. The introductory part is followed by the concepts part. In the concepts part there will be an extensive explanation of the path following algorithm used as it’s a core function. A solid understanding will aid the conviction that excluding certain common behaviors are legitimate for the purpose of comparing the two steering algorithms. There’s also sections for other behaviors such as steering and separation. In conclusion either of the steering algorithms\(^1\) could be used in a very crowded path traversal simulation.

This report was made independently and without collaboration.

---

1 I refer to them as steering algorithms, but in the report they are presented as seeking algorithms used for steering purposes.
# Table of Contents

- **Introduction** ............................................................................................................. 1  
  - Background .................................................................................................................. 1  
  - Goal ............................................................................................................................... 1  
  - Methodology ............................................................................................................... 1  
  - Test cases ................................................................................................................... 2  
- **Concepts** ...................................................................................................................... 2  
  - Path following ............................................................................................................. 2  
  - Seek ............................................................................................................................... 4  
  - Path following – improved path .................................................................................. 5  
  - Separation ................................................................................................................... 6  
- **Testing** ........................................................................................................................ 7  
  - Results .......................................................................................................................... 7  
    - Test I & II Standard paths for the steering algorithms ............................................... 7  
    - Test III – 100 boids ................................................................................................... 8  
    - Test IV – 300 boids ................................................................................................... 9  
  - Discussion and conclusion ......................................................................................... 10  
    - Test I & Test II – 1 boid & 10 boids ....................................................................... 10  
    - Test III – 100 boids ............................................................................................... 10  
    - Test IV – 300 boids ............................................................................................... 10  
    - Conclusion ............................................................................................................... 11  
- **References** .................................................................................................................. 12
Introduction

Background
A traditional basic flocking simulation consists of three core behaviors, cohesion, alignment and separation. Craig Reynolds, the inventor of the first flocking algorithm called the individuals in a flock boids so will I. Each boid moves towards the arithmetic average of all the other boids’ position. This behavior is called cohesion. Alignment behavior entails that the boid aligns itself in the direction of the other boids’ direction. Lastly the boid has a separation property which means that the boid separates itself from boids that are too close. These three simple rules are sufficient to create an accurate simulation of flocking behavior. Obviously there are many more behaviors that a flock may be capable of. Delightfully these behaviors can simply be added to the boid’s list of rules to enhance the simulation. [1]

Goal
This report is targeting crowding through narrow spaces. Which means flock traversal through narrow paths. The goal of the report is to examine and compare two different steering algorithms for path following, specifically crowd path following. Though it is possible to add behaviors to the three core rules as mentioned in the background this is not the procedure taken in this report. Instead cohesion and alignment behavior has been detached so only the separation behavior from the core rules are kept. The reason for this is that path following already specifies a direction for the boids to travel which makes following the majority of the boids redundant.

Given a target position both steering algorithms will end up at that goal. The difference lies in how they steer towards that goal. It is this difference that's going to be examined.

Methodology
Various tests are made on the different steering algorithms to examine its behavior. To begin with there is a need to observe how the steering algorithms acts while not in a crowd which will give a clear distinction between the different steering behaviors. The next natural step is to introduce more boids into the world, creating small crowds and eventually introducing a large number of boids creating one big crowd.

There will not simply be two comparisons per test case due to the fact that one of the steering algorithms is more flexible than the other. Therefore it is reasonable to test this algorithm with different values on the altering variable.

Other necessities for testing are:
1. **Path following.** Follows a predefined path that the boids has to follow. [2]
2. **Separation.** Boids separates themselves from other boids if they’re too close. [1]

---

2 Cohesion, alignment, separation
Test cases
There are five tests. Each of the tests differs in the number of boids.

- Test I: 1 Boid
- Test II: 10 Boids
- Test III: 100 Boids
- Test IV: 300 Boids

Concepts

Path following
To be able to follow a path, a predefined path must exist. The most basic path could be defined as a line between two points in a two-dimensional vector space. A simple path class could be defined as below:

```java
class Path{
    Vector2D start;
    Vector2D end;
    float radius;
}
```

The `start` variable defines the starting point of the path in a two-dimensional vector space and the `end` variable defines the end point. The radius decides the path’s width. See fig 1. A larger radius allows the boid to move further from the center and a smaller radius forces the boid to move closer to the center. Logically a wider radius will reduce the risk of collisions since there's more space to avoid each other in.

![Fig 1. A simple path.](image)

With a path defined the next step is to define the boid’s behavior in regard to the path. See fig 2. The behavior used is that of Craig Reynolds’ algorithm for path following[2]. By predicting the future location of the boid it is possible to determine if the boid will be outside the path radius. This is done by calculating the distance between the future position and the path, which happens to be a normal vector from the predicted location to the path. [3]
Fig 2. A simple path with a normal vector between the path and the predicted location of the boid.

The normal can be calculated by trigonometry. See fig 3. Since we are aware of the value of the start point of the path and the point of the predicted location we can construct a vector A between those points. With the knowledge of theta's value it’s a simple task to calculate the normal points.

\[
\text{calc_normal:} \\
\text{theta ← getAngle(A,B)} \\
\text{dist_start_to_normal ← A.magnitude() \times \cos(\text{theta})} \\
\text{B.normalize()} \\
\text{B.mult(dist_start_to_normal)} \\
\text{normal_point ← path.start.add(B)}
\]

Start by calculating theta, which is the angle between A and B. Then calculate the distance between the start point to the normal. Normalize B, where B is the path vector. Scale B to the distance between start and normal. Finally to get the normal points add B to the starting point of the path.[3]

Fig 3. Figure describing how to calculate the normal points.
The calculation of the normal vector can be simplified by using the definition of dot product. Which is:

\[ A \cdot B = |A| |B| \cos (\Theta) \]

The length of a normalized vector is one. Therefore if we normalize B, the dot product of A and B can be calculated using the following equation:

\[ A \cdot B = |A| \cos (\Theta) \]

This is the scalar projection of A onto B. So by taking the dot product into consideration the normal vector calculation can be improved by scalar projection.

```python
calc_normal(B, A):
    B.normalize()
    B.mult(A.dot(B))
    normal_point ← path.start.add(B)
    return normal_point
```

Hence the algorithm for path following looks like this:

```python
for each timestep:
    predicted_future ← predict_future(boid)
    normal_point ← calc_normal(B, A)
    distance ← predictLoc.distance(normalPoint)
    if(distance > path.radius):
        B.normalize()
        B.mult(x)
        target ← normalpoint.add(B)
        seek(target)
    else:
        keep follow path
```

If the distance from the predicted location to the normal point is larger than the path radius, the boid has to steer towards the path. If not, the boid is already following the path. According to Craig Reynolds the target is a point on the path a bit further from the normal point. See fig 4. So we normalize vector B and scale it to an arbitrary value x. Then we add B to the normal point and stores it into target, target is now a point x pixels ahead of the normal point. [3]

**Seek**

![Fig 4. Shows that when the boid is outside the path it will steer towards the path.](image-url)
craig_reynolds_seek(target):
    desired ← target.sub(pos)
    desired.normalize()
    desired ← mult(desired, speed)
    steer ← desired.sub(vel)
    return steer

A vector pointing from the boid’s position to the target position is stored into a variable \textit{desired} and scaled to a maximum speed. Finally the steer vector is calculated by subtracting the boid’s velocity from the \textit{desired} vector. [2]

conrad_parker_seek(target):
    desired ← target.sub(pos)
    steer ← divide(desired, x)
    return steer

Conrad Parker's algorithm is based on Craig Reynolds' which explains the similarity. As \textit{craig_reynolds_seek}, \textit{conrad_parker_seek} stores a vector pointing from the boid's position to the target position in a variable \textit{desired}. But then the algorithm divides the \textit{desired} vector with an arbitrary integer \(x\). If \(x\) equals to 100 the boid moves one percent towards the target.[4]

**Path following - improved path**

A path that’s defined by only a start and end point limits how the path can look like greatly. Which suppresses the usefulness of this study. To maximize the trustworthiness between comparisons of the steering algorithms it's necessary to have a path definition which enables flexible path constructions. Different paths can supply more test cases. By defining a path to be connected lines instead of just one line satisfies this.

```java
class path{
    ArrayList<Vector2D> points;
    Float radius;
}
```

Calculation for the target could be made by finding the normal vector in the former definition of paths. That method is not as easily applicable for the latter definition since a path does not consist of one single line. A path has several lines, which means that a path has several normal points. See fig 5. The need to identify the correct normal vector arises. Two conditions must be met by the normal vector to qualify as the correct normal. To even be considered, the normal point must be on the path. If they’re several normal vectors on the path, condition two chooses the correct one.[3]

1) The normal vector must be on the path
2) The closest vector is the correct vector
As previously stated, since the path consists of multiple lines we have to find the normal for all of them. We also have to check if the normal point is between the start and end of the line. If there’s no point on the path we use the end point of the line as a normal point. If we’re at the end of the line we want to specify a target point on the next line.[3] The latter options are not the case in this figure. The thick lines represents a rectangular pathway in which the circle formed boid moves close to. The thin lines are the possible normal vectors. By checking the conditions it's easy to determine which normal vector that's the correct one. Two vectors can be eliminated since they are not on the path and a comparison between the distance of the remaining vectors decides which is correct.

**Separation**

```plaintext
separate(thisBoid):
  for each boid:
    if (boid != thisBoid):
      if (|boid.pos - thisBoid.pos|< boid.radius*2):
        sep_vel ← sep_vel - (boid.pos-thisBoid.pos)
  return sep_vel
```

For each boid check if there is any boids that are too close. If they are, double the distance between them. By doubling the distance between the boids it will be possible for the boids to be inside each other for a brief period of time. See fig 6.

However since this code is not exclusively applied to one boid, but to every boid it will result in a smooth repulsion. [4]
Testing
There are four tests. The difference between the tests are that they’re two different algorithms for each test: craig_reynolds_seek and conrad_parker_seek. The latter is divided into four parts with different speeds against the target. The speeds are 100%, 10%, 1% and 0.1%. This means that there are five tests in total for every test case using craig_reynolds_seek and conrad_parker_seek with the four different speeds.

1. **Test I**: 1 boid moves through a path without separation.
2. **Test II**: 10 boids moves through a path without separation.
3. **Test III**: 100 boids moves through a path with separation.
4. **Test IV**: 300 boids moves through a path with separation.

Results

**Test I & II Standard paths for the steering algorithms**
Since test I and test II yields the exact same result only illustrative data from test I is presented. The illustrations represents a lap of a boid. Videos from where the illustrations where originated from are also provided.

---

3craig_reynolds_seek
4conrad_parker_seek, 100% towards the goal
5conrad_parker_seek, 10% towards the goal
6conrad_parker_seek, 1% towards the goal
Note that in the videos some boids are intersecting with each other in test II. This is a non-issue due to the fact that the test was designed to study the steering behavior exclusively therefore separation behavior is disregarded.

**Test III - 100 boids**

All the illustrations from here on show a random lap of a random boid. The rest of the boids are made invisible to enhance observation. To view the full interaction between the boids videos are also provided.

---

7 conrad_parker_seek, 0.1% towards the goal
8 craig_reynolds_seek
9 conrad_parker_seek, 100% towards the goal
10 conrad_parker_seek, 10% towards the goal
11 conrad_parker_seek, 1% towards the goal
**Test IV - 300 boids**

12 conrad_parker_seek, 0.1% towards the goal
13 craig_reynolds_seek
14 conrad_parker_seek, 100% towards target
15 conrad_parker_seek, 10% towards target
16 conrad_parker_seek, 1% towards target
Discussion and conclusion

Test I & Test II - 1 boid & 10 boids
The results from test I indicates that there are ideal traversal paths for the boids. Test II supports this conclusion since all boids follows the same path pattern after an arbitrary amount of time which can be seen in the videos of test II. By observation of the two first illustrations we can conclude that Conrad Parker's seek with 100% steering is equivalent to Craig Reynolds' seek algorithm. See footnote three and four.

Conrad Parker's seek algorithm behaves differently depending on what weight is given to the steering. 100% steering causes the boid to cover the least amount of space. As the steering loosens we can observe that the boid covers more space. At 10%, see footnote five, a relatively major change in the ideal path is made by the boid and as the steering loosens the boid moves more and more alongside the outer wall of the path.

Test III - 100 boids
As concluded from test I and test II Conrad Parker's seek with 100% steering is equivalent to Craig Reynolds's seek hence test III should produce similar results using these algorithms. Illustration one and two of test III shows that this is indeed the case. See footnote eight and nine. Illustration three, see footnote 10, is also similar to one and two, but as the steering loosens the shape of the lap changes unpredictably.

A conclusion can be made, the weaker the steering is the more unpredictable the boid's traversal route is. Another factor for the unpredictability is due to the limitation of path population to 100 boids. This limitation creates a sparse path, which allows the boids to move relatively freely.

Test IV - 300 boids
Illustration one to four, see footnote 13 to 16, are quite similar. The reason for this is that because the path is inhabited by 300 boids there is no free space to move around in as clearly shown in the videos of test case four. The boid moves in a more or less straight line. However, the last test case show a irregular, unpredictable traversal. This is a result of the extremely weak steering combined with the relatively overwhelmingly powerful separation velocity. Since the magnitude of the separation velocity

17conrad_parker_seek, 0.1% towards target
is much larger than the steering velocity the boids separates themselves from each other heavily away from the path.

**Conclusion**

In conclusion either of the seek algorithms could be used in a very crowded path traversal simulation for steering. But, the steering weight of conrad_parkers algorithm must be fairly strong. Test case four suggests that the steering must be at least above 10%. However I would propose using Conrad Parker's algorithm for it's adjustable steering velocity. Parker's algorithm can easily be adapted to produce the desired boid behavior in regard to the path's population.
References