The Krillians

Jules Bachmann, Anna Jaeggi, Felix Sarnthein, Laura Wülfroth

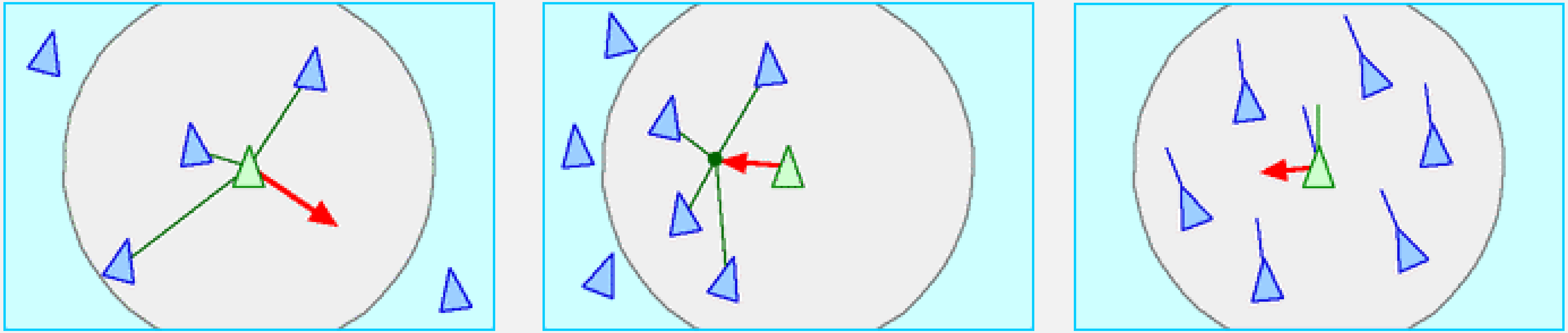
Our goals

- Implementation of a krill swarm
- Analyse the swarm reacting on escaping krill

  → After what percentage of escapists does the swarm follow?

Boids model

Force Components: Separation, Cohesion, Alignment



The escapists

- Move towards target vector
- Still influenced by other krills

Implementation

Autonomous Character Model by Craig Reynolds

- agent-based model for animations
- autonomous individuals follow a set of rules
- restricted perception of environment

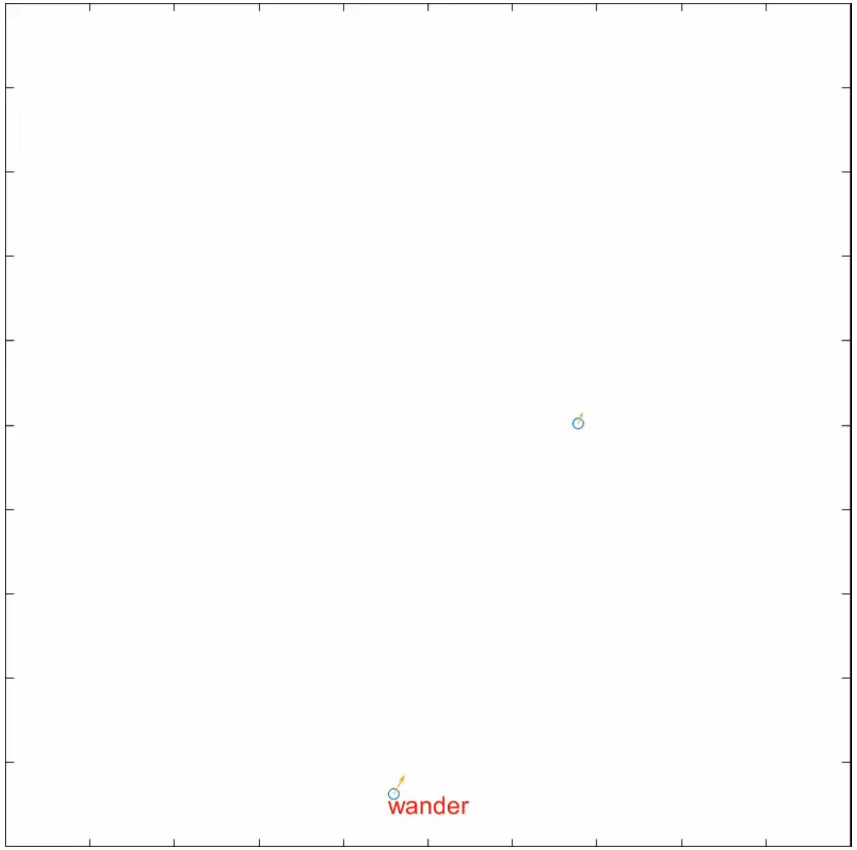
➢ impression of reproducing a behaviour

Implementation

Non-Social Behaviours

- *seek* a target
- *flee* a target
- *wander* randomly but naturally → seek a random target in front

➢ individuals respond to a target



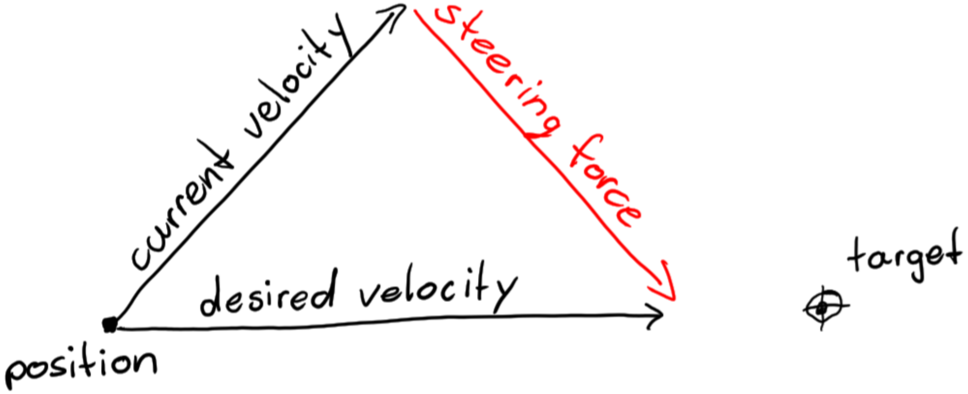
Implementation

The Force Model

- key concept
- makes animations feel natural
- allows for modular approach

➢ multiple forces are added together

➢ agility is controllable



```
steering_force = desired_velocity - current_velocity

current_velocity = current_velocity + steering_force

position = position + current_velocity
```

Implementation

But most importantly:
Usefull abstraction of steering behaviour

→ every behaviour only needs to specify its desires

Implementation

Non-Social Behaviours within the Force Model

behaviour_x	**computeDesire(…)**
seek	`desired_velocity = + (target – position);`
flee	`desired_velocity = – (target – position);`
wander	`desired_velocity = + (target* - position);`

```
steering_force = behavior_x( position, velocity, environment){

  desired_velocity = computeDesire(...);
  steering_force = computeForce(velocity, desired_velocity);
  return steering_force;

}
```

Implementation

Social Behaviours within the Force Model

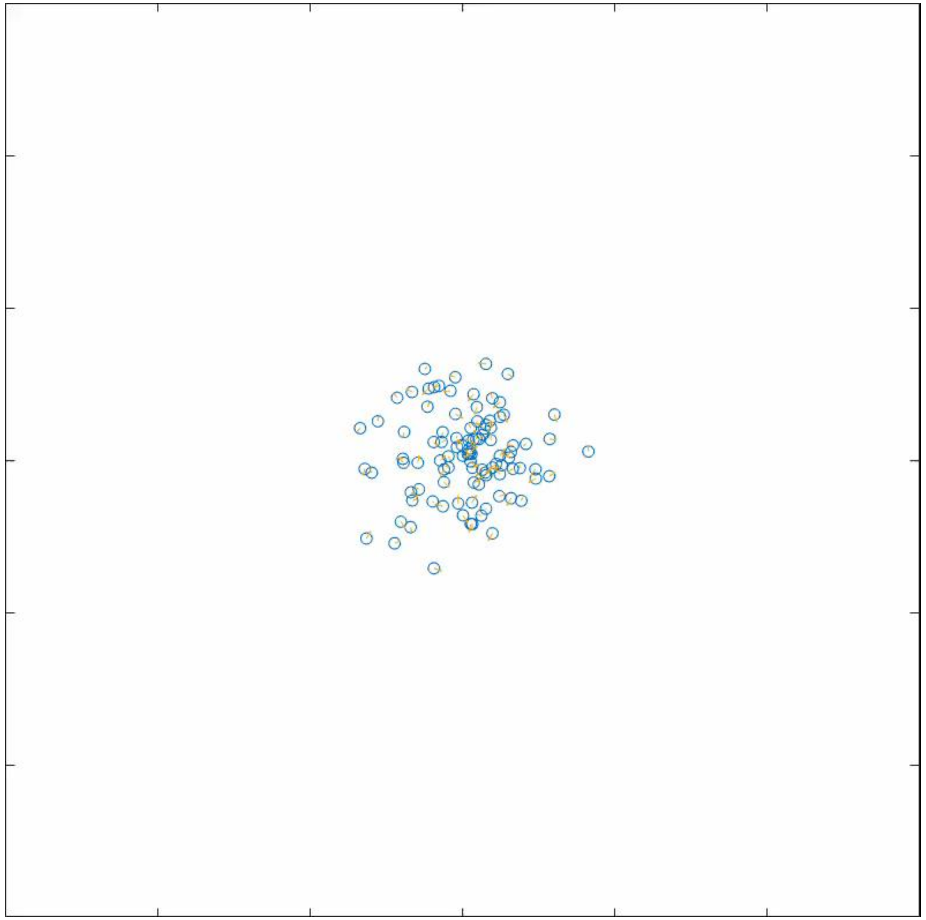
behaviour_x	**computeDesire(…)**
cohesion	`desire = + (neighbor - position);`

separation	`desire = - (neighbor - position);`
	`desire = desire / norm(desire)^2;`

alignment	`desire = neighbor_velocity;`

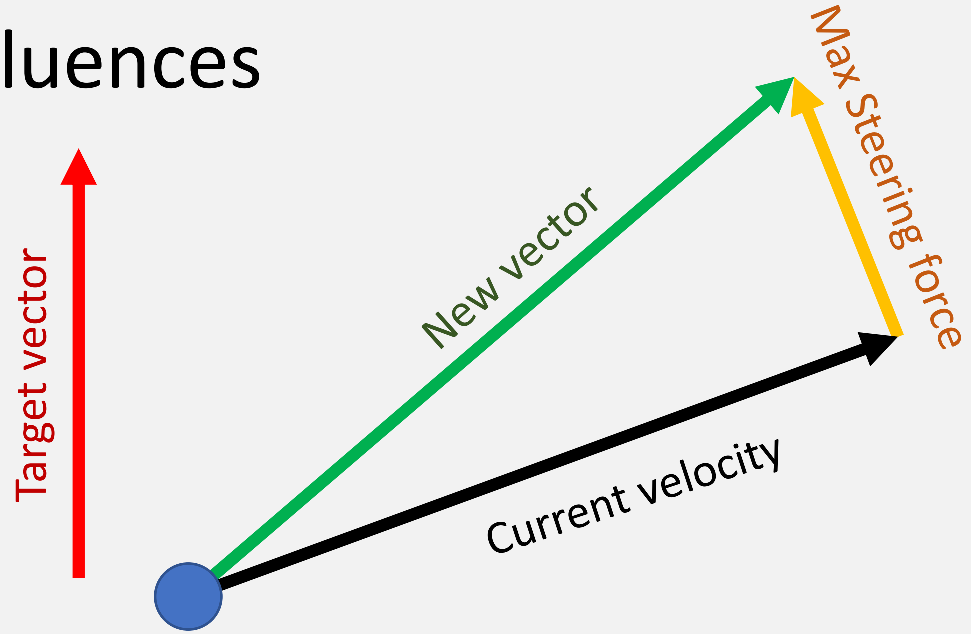
```
neighbors = rangesearch( positions, radius);

for all neighbors i
  desires[i] = computeDesire(...);
desired_velocity = mean(desires);
```

Implementation



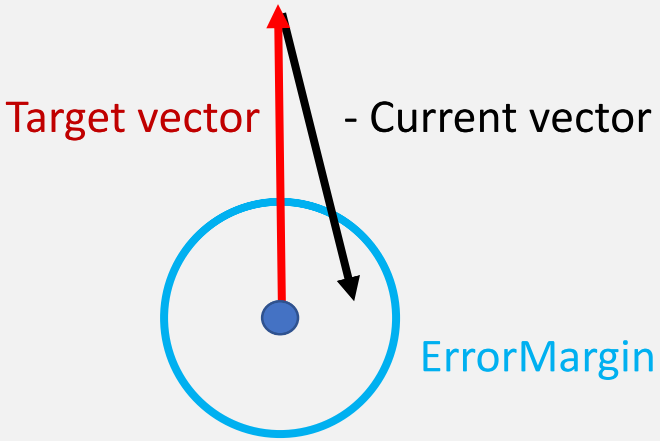
Implementation of the escapists

- Vector A: target vector and steering force
- Combinate with normal influences

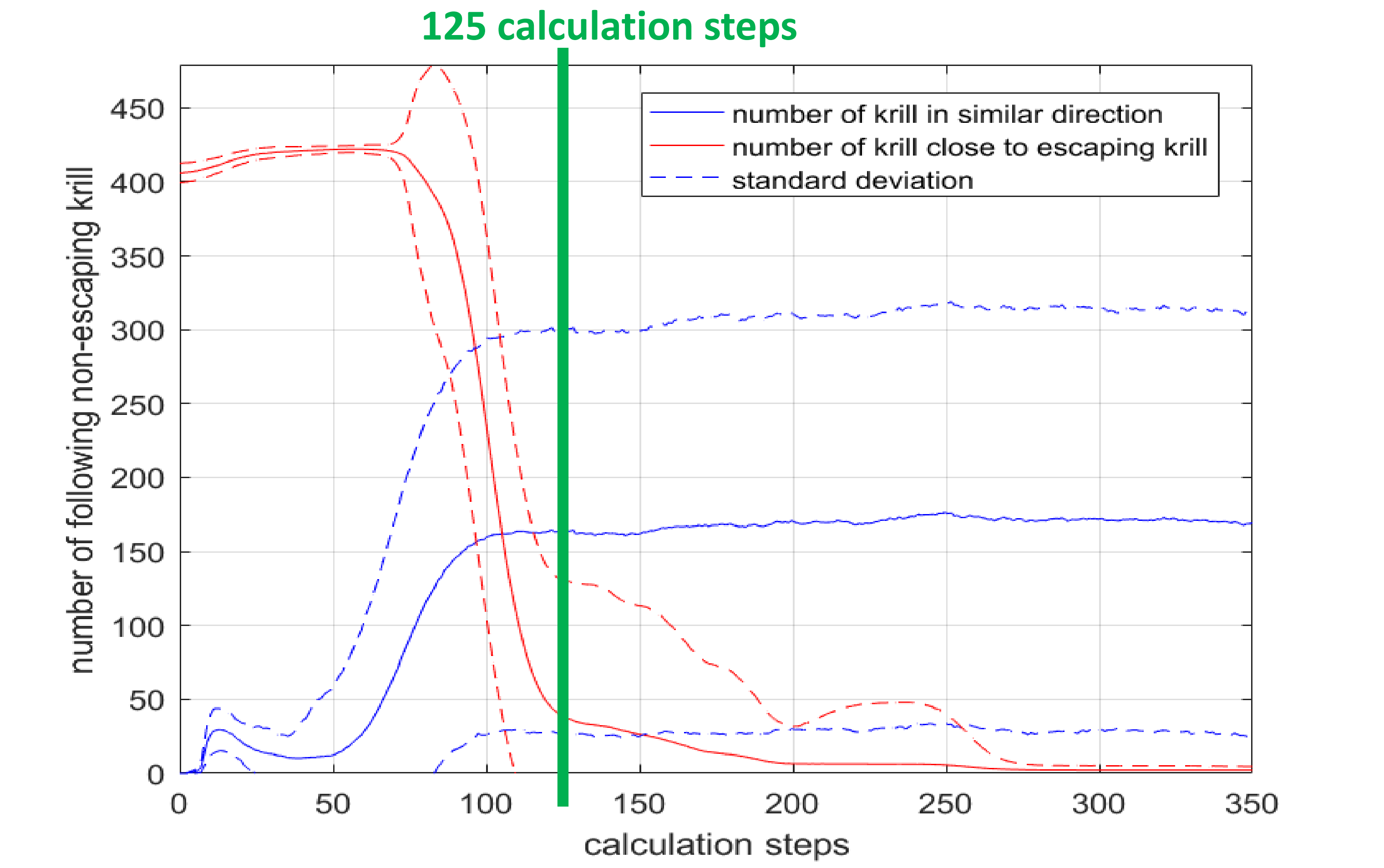


Computation of the results

- Following target vector
- Closeness to escapists
- Computation every single simulation step

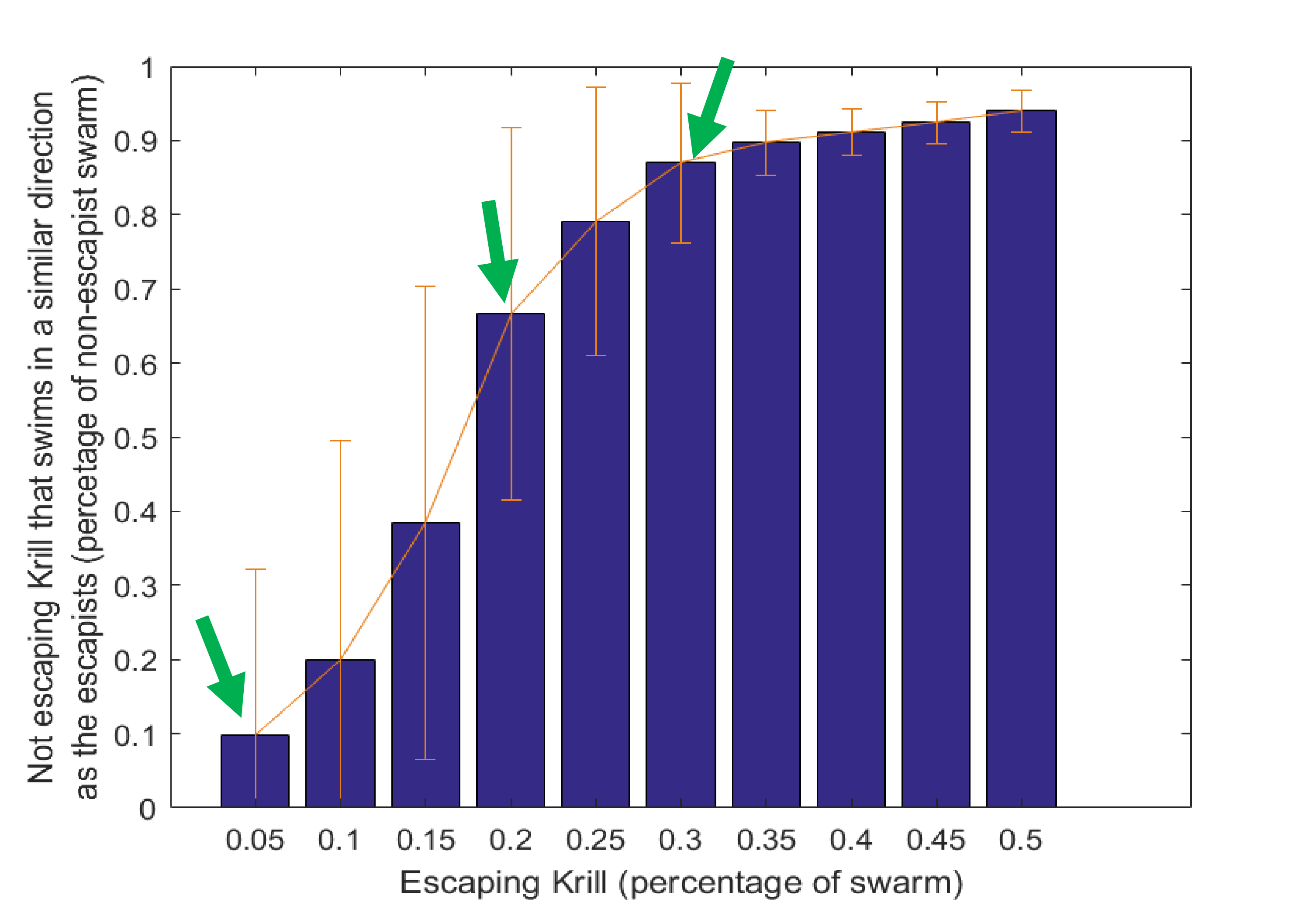


Results

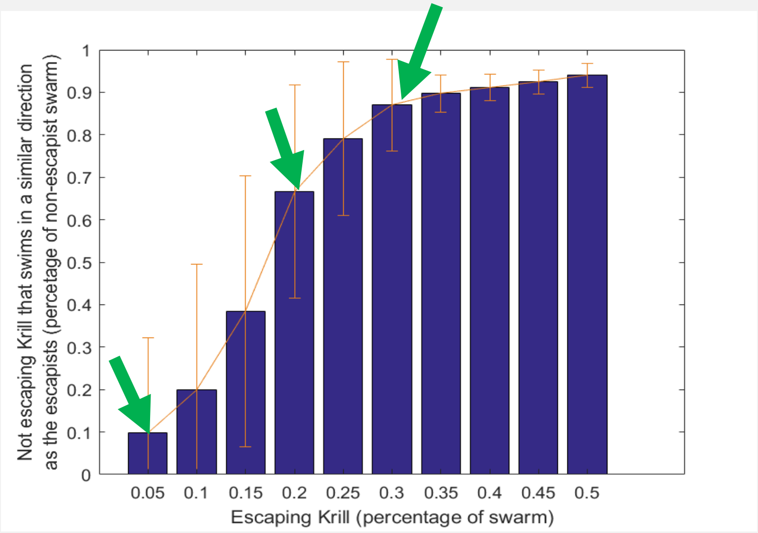

125 calculation steps
number of following non-escaping krill
0
50
100
150
200
250
300
350
400
450
calculation steps
number of krill in similar direction
number of krill close to escaping krill
standard deviation

Results of 15% escaping krill and a target vector [0,1]

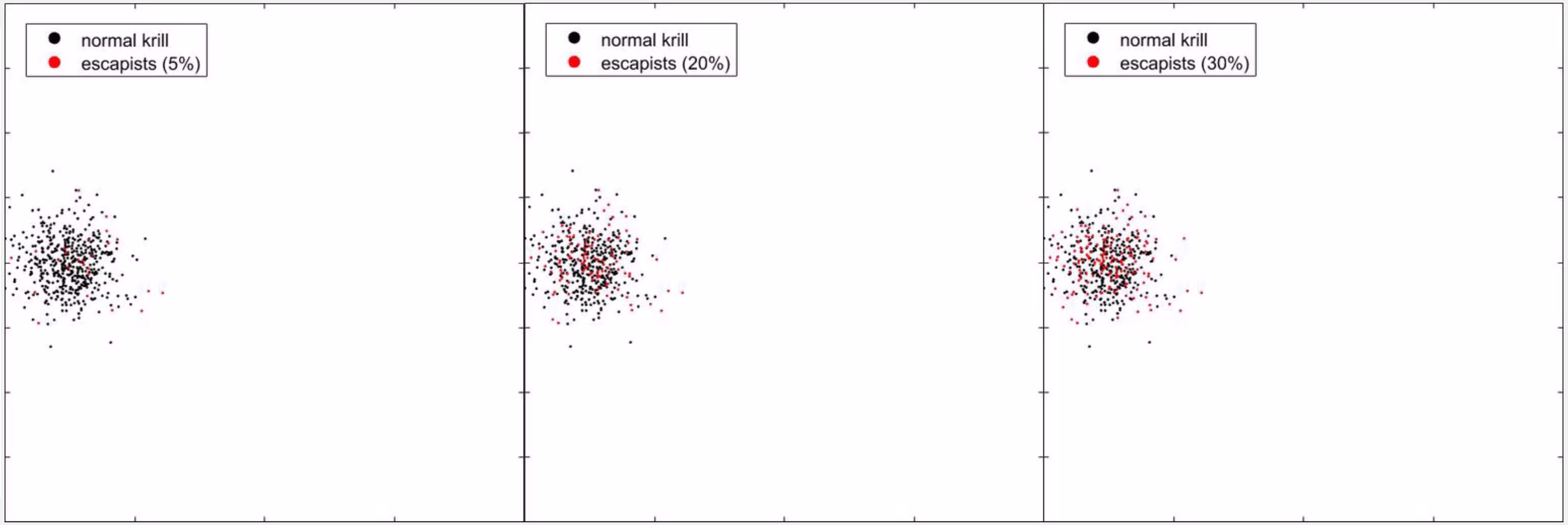
Results


Not escaping Krill that swims in a similar direction
as the escapists (percetage of non-escapist swarm)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Escaping Krill (percentage of swarm)

Results

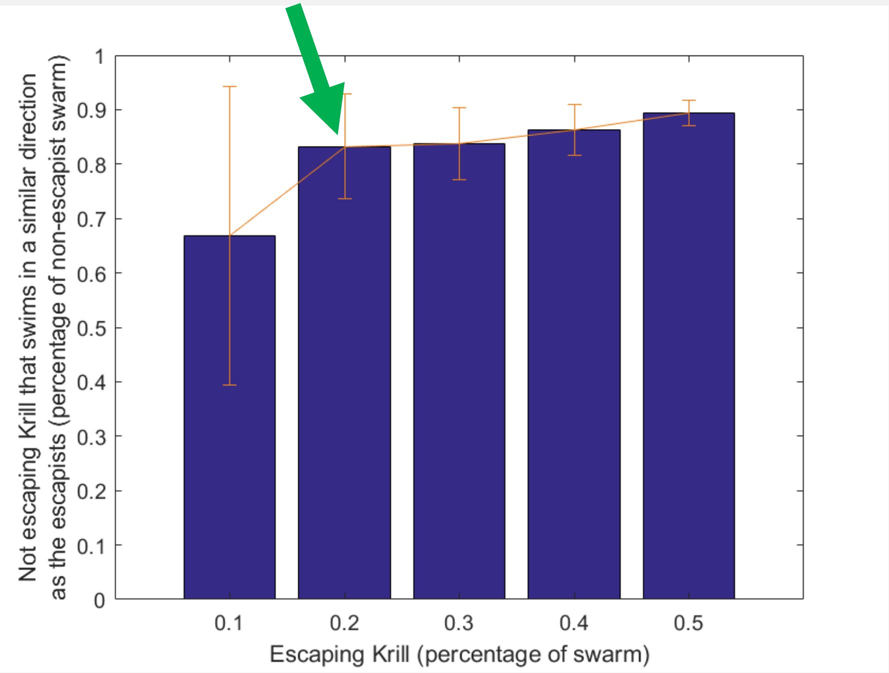


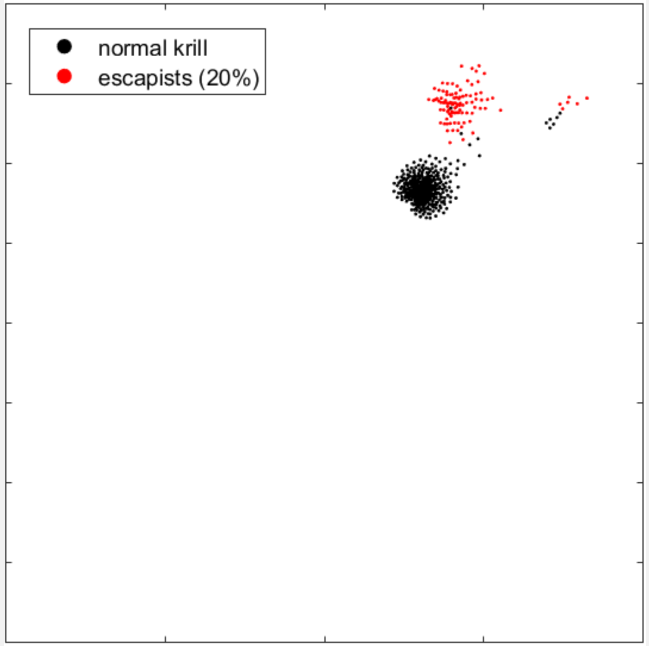
Not escaping Krill that swims in a similar direction as the escapists (percetage of non-escapist swarm)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
Escaping Krill (percentage of swarm)



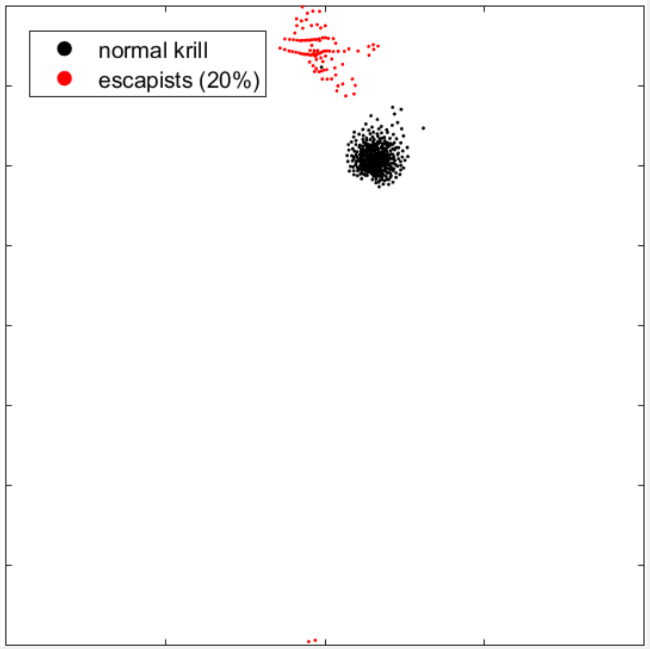
● normal krill
● escapists (5%)
● normal krill
● escapists (20%)
● normal krill
● escapists (30%)

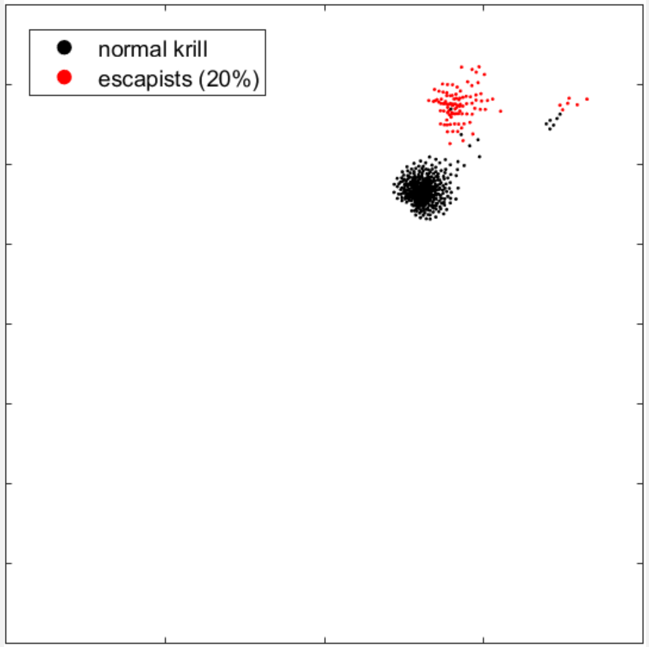
Results


Not escaping Krill that swims in a similar direction as the escapists (percentage of non-escapist swarm)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
Escaping Krill (percentage of swarm)


● normal krill
● escapists (20%)

Results





Summary and Outlook

- After what percentage of escapists does the swarm follow?

  →30% for [0,1], ~20% for [1,1]

- Problem: “similar” vector (the range)
- Further simulations:
  - Swimming direction of the swarm
  - Position of the escapists
  - Current

Questions?