Leader-based flocking algorithm in 3-dimensional space

Felix Leon Griebel

02.12.2024

1 Introduction

Robotic swarms gained significant importance in the last years, as they can be used in many different ways. In rescue missions they could be used to cover a wide area of terrain. Or in underwater exploration they can be used, to model a good picture of the seabed. Coordinating these robots in a dynamic environment presents us with a challenge. At the same time, we want to minimize the collision, optimize the completion of the task, and ensure a good efficiency. Traditional approaches often rely on manual control of the individual robots, which can be prone to errors, is expensive in human resources and impractical for a large-scale setting.
Biologically inspired flocking algorithms have already proven a good swarm coordination by simulating the behavior of birds or fishes. These algorithms focus on decentralized control, where each member applies simple rules like alignment, cohesion, and separation. However, even though these methods are less complex and well established, they do not necessarily avoid obstacles other than the other swarm members. Additionally, they may form multiple swarms instead of a single one.
To address these limitations, machine learning (ML) came up as a promising tool to improve the coordination of a swarm. ML techniques can optimize the flocking parameters or predict collision scenarios. One example is Reinforcement Learning (RL) that allows the members of the swarm to learn and adapt based on their actions in the environment.
The goal in this project is to implement a flocking algorithm that enables real-time control or a robotic swarm while introducing the human resources needed.
The Boids algorithm will be used as a foundation. Additionally, we will explore the use of RL in combination with the developed environment and evaluate the findings.

2 Related Works

There are several approaches that implement a similar idea.
First, one study focuses on flocking in underwater environments, where a single leader is introduced to guide the swarm as target while avoiding collisions. They validate the approach by using MATLAB simulations[Kim23].
Another project explored flocking in quadcopters, inspired by biological examples to improve the swarm behavior while also exploring the factors influencing these natural flocking algorithms[VVT+13].
Recently, machine learning has been integrated into swarm robotics to tackle the limitations of traditional algorithms. One review presents various ML techniques, such as reinforcement learning and clustering methods, to improve the flocking capabilities of unmanned aerial vehicles (UAVs), focusing on problems like dynamic obstacle avoidance and communication inefficiencies[AHR21].
Similarly, another paper explores the use of virtual leaders in swarm control. Based on an existing algorithm, modifications are introduced to allow a more flexible and scalable approach[SWL09].
These advancements show the potential of combining traditional flocking algorithms with machine learning to improve adaptability in an environment with additional constraints, e.g., such as obstacle avoidance and leader control. In addition, other valuable resources include the informational presentation of a Boids algorithm with obstacle avoidance, simulated in the Unity engine[Lag19] and the description of the conventional Boids algorithm[Ada20].

3 Methods

3.1 Environment

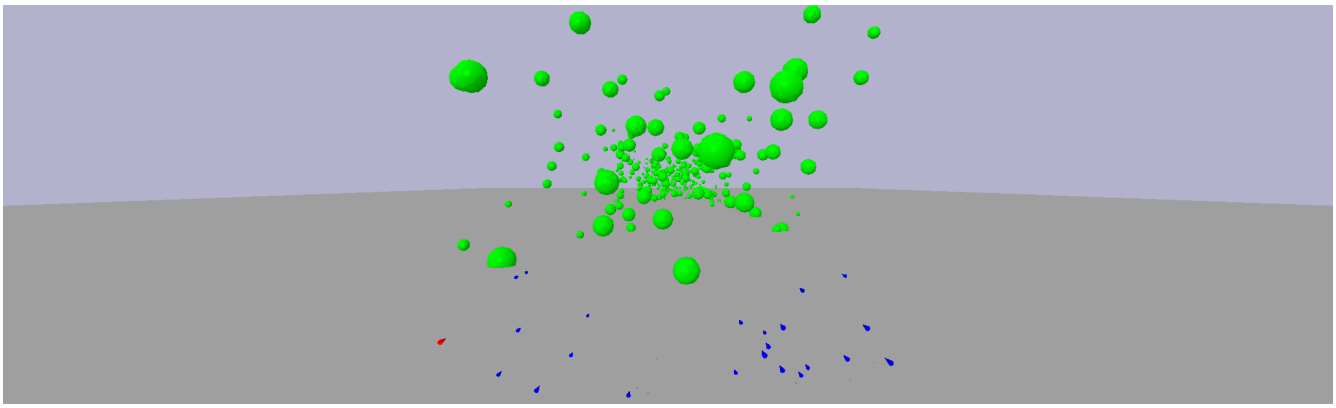


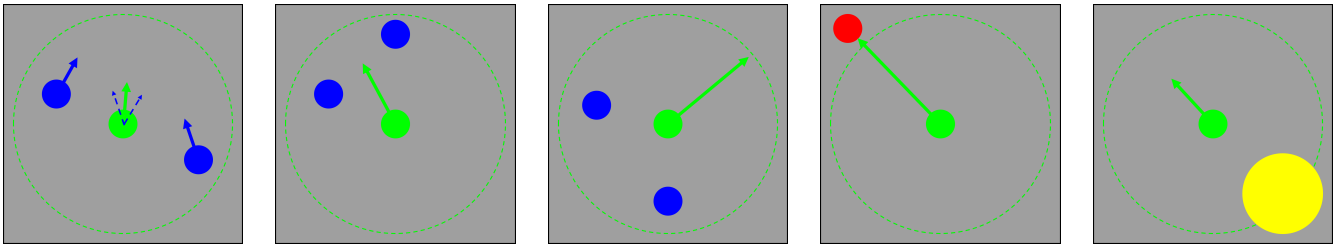
Figure 1: The simulation environment, drones ins blue (members) and red (leader). Spherical obstacles in green.

For the simulation environment, we use the PyBullet physics engine, giving us a robust platform to model our swarm in three-dimensional space with real-time interactions and a graphical user interface when needed.
In the environment, we initialize 30 drones, represented by cones, which will be the members of our swarm. The first one will be the leader, marked in a different color than the other drones. The drones are randomly distributed in an area above the origin of our simulation environment.
The leader drone is meant to be controlled by the user, using the keyboard. However, to ensure a more reproducible training, we recorded user interaction in the environment that the leader drone can follow. The other members of the swarm will behave based on different, implemented approaches.
Additionally, the simulation environment will be filled with obstacles of various sizes and positions. These obstacles will introduce a new complexity to the task that the drones will have to handle in addition to following the leader, and keeping up the swarm coordination. The obstacles will also be placed in fixed positions to ensure deterministic training and testing stages. We see an example of a simulation environment in figure 1 with obstacles marked in green and drones in red and blue.
The movement of the drones is only limited by the path they can travel in each frame of the simulation. So no matter the model, they are not supposed to stand still or go too fast. The directions in which they move each update is not limited. This is limited for the user-controlled leader drone though, as it is controlled by changing the yaw and pitch angles of the object. This descision was made, to simplify the control for the user.
In each simulation in our approaches, the leader drone will travel a specific path, e.g., to the other side of the area containing the obstacles. If the leader drone is done with this path, the simulation will stop. In case members of the swarm collide during the simulation, they will be removed from the environment.

3.2 Basic Algorithm

We first implemented a basic algorithm as a baseline, taking inspiration from the Boids algorithm. However, we make some changes and adjustments to it.
As in the traditional flocking algorithm, we have three important principles: Alignment, Cohesion and Separation. Alignment matches the direction of closeby drones for a unified direction of the swarm. Cohesion is used to pull the swarm together, so that the members do not drift off, and Separation aims to prevent collisions between the drones. In figure 3 we can see examples of different combinations of separation and cohesion
All of these principles can be represented as vectors that can be combined into a single one. To combine them, weights are used to scale the different vectors before combination. These principles are usually calculated for each drone in respect of the drones in specific radii around it. We will use only one radius for all of the principles to improve the performance.
In order to introduce the leader to the swarm, we additionally add the leader-follow principle and include the direction vector to the leader in the combination. Furthermore, we will also define an obstacle-avoidance vector by keeping track of the obstacles in a drone's perception radius.

The result of these five principles is a weighted sum that will additionally be normalized, thus presenting a new movement vector for a drone. The normalization ensures a consistent speed as previously mentioned. For a better understanding, the principles are visualized in figure 2.
This approach is based on traditional flocking algorithms with some adjustments. However, the main limitation in this approach is the definition of the weights for the combination of the different parameters. Defining the weights for the calculation ends up in a trade-off between the different goals of the principles. Furthermore, the importance of the different principles may differ depending on the current situation. For example, if an obstacle is close to a drone, the avoidance of this obstacle is more important than following the leader in that situation.



(a) **Alignment** aligns the movement of neighbours, improving a synchonized movement.

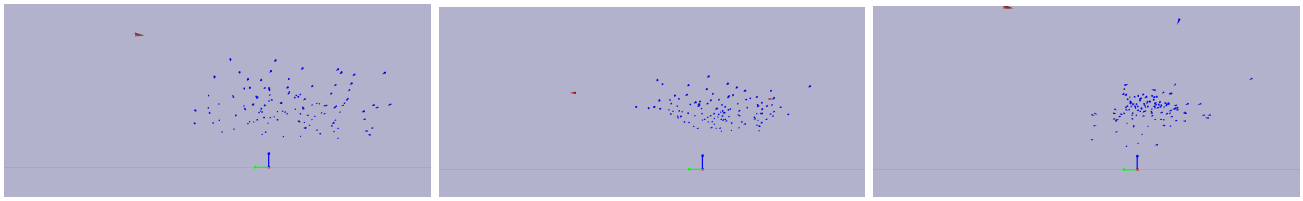
(b) **Cohesion** encourages a drone to move towards the center of a group of drones

(c) **Separation** prevent members from crowding and helps keep a distance to the neighbors, avoiding collisions.

(d) **Leader-following** ensures that members follow the target, even if it is outside of the perception radius.

(e) **Obstacle avoidance** help the member keep a distance to non-drone obstacles, avoiding collisions.

Figure 2: The five principles visualized for the current drone (gree) in interaction with other members (blue), the leader (red) and spherical obstacles (yellow). In addition the perception radius is represented as a dotted, green line.



(a) Swarm with high separation and low cohesion.

(b) Swarm with lower separation and medium cohesion.

(c) Swarm with very low separation and high cohesion.

Figure 3: Drone simulations with differing density by changing the parameters for boids algorithm.
The factor for leader-following is close to 0.

3.3 Centralized RL approach for weights

The first idea was to introduce a reinforcement learning model, learning and predicting the weights of the principles for the whole swarm. That way the weights for the five principles would be updated before all drones calculate their new movement vector for a frame. However, this approach has many limitations and yielded bad results, as to why we will not discuss it in depth.
Firstly, having the same weights for all drones gives us just a slight advantage over the basic approach. However, the situations of each drone differ usually, meaning that it is a disadvantage, still having the same weight across all swarm members. Secondly, in a real-world scenario, this approach would result in a strong centralized component, increasing the need for communication.
Lastly, another problem is the observation space used for the RL model. For this approach, the observation space would most likely have many dimensions. Besides these limitations, this approach is also extremely inefficient in terms of learning, as the model iterated over all drones in each step.

3.4 Decentralized RL approach for weights

As a resulting idea, we focus on predicting the weights for each drone separately. This results in quite few advantages over the centralized approach.
Firstly, the model could adjust to a drone's specific situation. And secondly, we can utilize the update of each drone separately in the learning process. This means that we would yield an increase of steps per iteration of the drones by a factor of 30 in comparison to the centralized approach (in an optimal run if no drones collide).
As an observation space for the model, we simply use the already calculated vectors. We need to calculate them for the combination either way. Additionally, this will also have an extra advantage, as we will discuss in the following approaches.
The result of the model's prediction is five factors between 0.0001 and 1.0.
This approach promises us advantages over the baseline algorithm and the centralized approach for weights. We try to improve the adaptability of the movement to specific situations, while still having the basis of conventional flocking.
However, the outcome may depend strongly on the quality and diversity of the training of the model.
Additionally, the outcome is still heavily influenced by the base principles.

3.5 RL approach for movement vector I

The previous approaches discussed the prediction of the weights to combine the different vectors of the respective principles.
We now want to explore further by removing the direct influence of these principles. The idea is to predict the movement vector directly. One possible advantage may be a better performance, as we do not need to calculate the different vectors.
While we remove the principle ideas in this approach, we still focus on making predictions for each drone separately, as it improves the model's training efficiency and is only logical for predicting movement vectors.
For the observation space of the model, we combine the positions and distances to different obstacles and drones within the perception radius of a drone. We also include the position of the leader and the distance to it and the drone's current velocity. We also limit the neighbors and obstacles to 10 and 5, respectively. The result is again a high-dimensional observation space, which may lead to limitations of this approach. High dimension typically lead to a worse representation of states in the training process, which leads to limited prediction abilities. Another problem is that the observation space is normally dynamic. Even though we try to limit the upper bound of the close drones and obstacles, we can not ensure that we always have these numbers in the perception radius. Thus, we need to add padding to the observation spaces, which may lead to misinterpretations.
The prediction result will be a vector of length three with values between -1.0 and 1.0 . This configuration would allow the model to predict the zero vector, which is undesirable for us. Thus we have to remove this possibility.
The limitations of this approach definitely include the high-dimensional and dynamic observation space. Even though we expected an improved efficiency, we now have to calculate the observation space previous to the prediction. Also, the model's prediction will directly translate to the movement of a drone, which may be an advantage but could also be a disadvantage if the output is not reliable.

3.6 RL approach for movement vector II

Lastly, we extend the previous approach for the movement vector. One big problem was the observation space, thus the main idea is to go back to using the 5 principles again. We use these principle vectors (except the alignment) as observation space for the model and receive a movement vector back. That way, we give the model more influence on the outcome. At the same time, we significantly decrease the size of the observation space and also remove the need for possibly padding the observation space with zeros to reach a specific size. The calculation of the five principle vectors is not significantly larger than the calculation of the observation parameter for the previous approach and should not result in a major disadvantage. The main limitation is here still the direct influence of the model, which depends on the training process.

4 Evaluation

4.1 Setup

To evaluate the different RL approaches, we build 3 models for each of them. One only trained in environments with fewer obstacles (marked as small), one trained in ones with a high number of obstacles (marked as big) and one trained in all types (marked as combi). The small option will be trained on environments with 0, 250 and 500 obstacles, the big option on ones with 1000, 1250 and 1500 obstacles, and the combi option on environments with obstacles counting from 0 to 1500 (in 250-steps). This different obstacle count corresponds to the complexity of the environment, meaning that more obstacles present the members more possibilities for collisions. Additionally, all models will be trained on 3 possible paths for the leader. This results in each model having between 9 and 21 different training iterations in different environments with 2048 steps, 20 epochs and 104856 total timesteps per iteration. We used these parameters to balance the training stability and convergence speed during training. Additionally, we tried to prevent overfitting with the chosen number of epochs.
The termination and truncated conditions, as well as the reward function in the training stage are for all RL approaches similar. The training will penalize collisions with other drones or objects highly, and also reduce the reward, the further the drone is away from the leader. One significant difference is that the second RL approach for the movement vector, the truncated condition includes the check of the average distance to the leader.

We will test the models on a new leader path with a medium number (750) of obstacles on the path.
To evaluate the different approaches and models, we will take a look at three different parameters: first, how many drones survived after a simulation execution, the average computation time needed to update one drone in milliseconds, and lastly the average distance to the leader over the simulation. The two latter parameters will be firstly averaged over all drones in each step and then averaged over all steps in one simulation execution.

4.2 Basic algorithm

For the basic algorithm, we used four different options with different weight values: option 1 with balanced weights, option 2 with a high lead vector, option 3 with high obstacle avoidance weighting, and option 4 for high cohesion. In table 1 we can see the specific weights for the options.

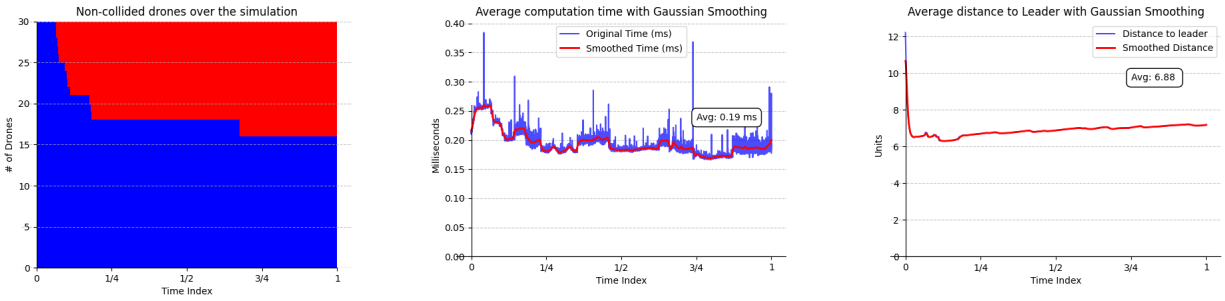
	Option 1	Option 2	Option 3	Option 4
Alignment	0.5	0.3	0.2	0.4
Cohesion	0.4	0.2	0.3	0.8
Separation	0.3	0.4	0.7	0.2
Lead-Following	0.6	0.9	0.4	0.5
Obstacle-Avoidance	0.7	0.6	0.9	0.6

Table 1: Weights for principle vectors.

After the execution of the tests, we observe unexpected results, visible in table 2. Even though the second option was optimized for a high leader following, the average distance is higher compared to the other runs and the option focusing on high obstacle avoidance still had more collisions than option 4 . The most promising results yield option 4. The results of option 4 are visible in more detail in figure 4. We see that the number of collisions mostly appears in the first quarter, possibly when the swarm first faces the obstacles. Once they move inside the field of spherical obstacles, the number of collisions is significantly decreased. Also, the distance to the leader stays relatively constant, which means that even in a longer run, the drones do not stay back.
However, the values might change depending on a different path or configuration of the obstacles, so the results may vary in other scenarios, and other options might be more suitable.

	Option 1	Option 2	Option 3	Option 4
Drone survival	10	10	12	**16**
Avg. update time	0.14	**0.13**	**0.13**	0.19
Avg. lead. distance	7.53	8.58	16.51	**6.88**

Table 2: Results for base algorithm tests.



(a) Survival of drones over the execution (number of alive drones in blue, number of collided in red).

(b) Average update times over the execution. Gaussian filter ($\sigma = 20$) applied in red.

(c) Average distance to the leader over the execution. Gaussian filter ($\sigma = 20$) applied in red.

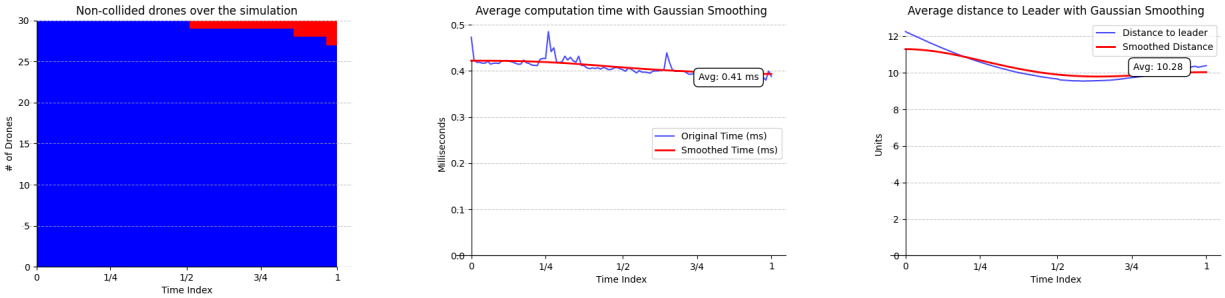
Figure 4: Graphics for the test results of the baseline algorithm option 4.

4.3 Decentralized RL approach for weights

The results for the three models can be seen in table 3. We can observe that the model trained only on environments with a small number of obstacles performed the best. The update time is relatively high, however, the value is also dependent on the number of drones in the environment, as the principle vectors that need to be calculated are calculated on the drones in the perception radius.
For a better insight, we also see the specific graphs for the test of the small-model in figure 5. We see that the number of surviving drones is extremely high compared to the baseline approaches. The distance to the leader is slightly higher, but still acceptable.

	Small	Big	Combi
Drone survival	**27**	4	7
Avg. update time	0.41	**0.23**	0.29
Avg. lead. distance	**10.28**	40.07	68.13

Table 3: Results for decentralized RL approach for weights.



(a) Survival of drones over the execution (number of alive drones in blue, number of collided in red).

(b) Average update times over the execution. Gaussian filter ($\sigma = 20$) applied in red.

(c) Average distance to the leader over the execution. Gaussian filter ($\sigma = 20$) applied in red.

Figure 5: Graphics for the test results of the decentralized RL approach for weights, only trained on enviropnments with low number of obstacles.

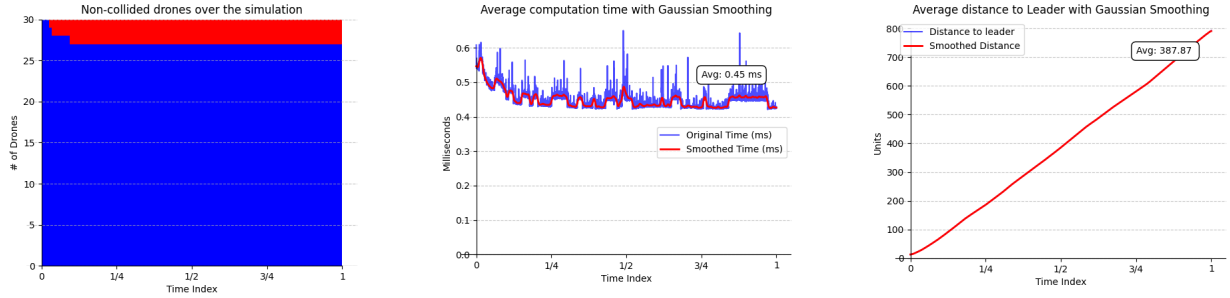
4.4 RL approach for movement vector I

In this approach we tried to predict the movement vector directly. However, when we look at the results presented in table 4, we see a significant problem. Although the number of surviving drones for all models is relatively high, the average distance to the leader is extremely high (for reference, the distance through the obstacle course is about

600.0 units). This suggests that during the training process, the model prioritizes low collisions and accepts a low reward in return.
While exploring this model, we tried to penalize a higher distance to the leader even more but could not achieve different outcomes.
For a better understanding, we can take a look at the graphics in figure 6a, which shows the results for the big-model.
The findings that the model prioritizes high leader distances over accepting more collisions, were used in the development for the second approach for the movement vector. This was the reason why we added the truncated-condition that a path is truncated if the average leader distance is too high.

	Small	Big	Combi
Drone survival	19	**27**	21
Avg. update time	0.4	0.45	**0.37**
Avg. lead. distance	**283.93**	387.87	484.95

Table 4: Results for RL approach for for the movement vector.



(a) Survival of drones over the execution (number of alive drones in blue, number of collided in red).

(b) Average update times over the execution. Gaussian filter ($\sigma = 20$) applied in red.

(c) Average distance to the leader over the execution. Gaussian filter ($\sigma = 20$) applied in red.

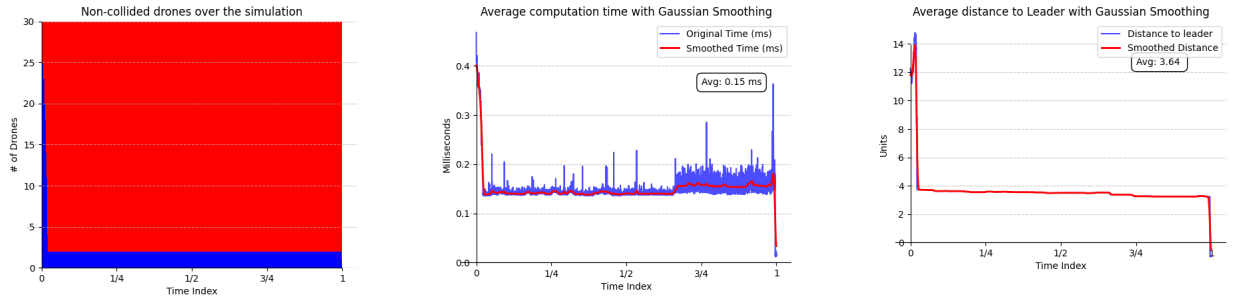
Figure 6: Graphics for the test results of the RL approach for the movement vector I, only trained on enviropnments with high number of obstacles.

4.5 RL approach for movement vector II

Lastly, we evaluated the results for the second RL approach for the movement vector below in table 5. The results may be confusing, so we will discuss the meaning behind the values shortly. The small- and big-models have surprisingly low update times and no average distance. This is the case, as the only drone surviving in the end is the leader drone, meaning that all model-coordinated members collided during the execution. In the combi-model we can make an interesting observation. If we were to look at the execution in a graphical interface, it would be visible that one surviving member learned to follow the direct path of the leader drone while keeping a distance to it. However, this last member crashes slightly before the end of the execution. This behavior is also visible in respective graphics in figure 7
The combination model shows an interesting case, which might suggest that with more learning rounds, a model could be able to produce executions where not all members collide and a low distance to the leader is held up.

	Small	Big	Combi
Drone survival	1	1	1
Avg. update time	0.02	0.13	0.15
Avg. lead. distance	-	-	-

Table 5: Results for second RL approach for for the movement vector.



(a) Survival of drones over the execution (number of alive drones in blue, number of collided in red).

(b) Average update times over the execution. Gaussian filter ($\sigma = 20$) applied in red.

(c) Average distance to the leader over the execution. Gaussian filter ($\sigma = 20$) applied in red.

Figure 7: Graphics for the test results of the RL approach for the movement vector II, trained on a combination of low-obstacle and high-obstacle environments.

5 Conclusion and Limitations

The presented approaches explore possible extensions for flocking algorithms for a swarm of drones. They focus on limitations of other approaches and try to solve them.
We first built a simulation environment and afterwards a baseline algorithm, focusing on principles of traditional flocking algorithms. Afterwards we tried to integrate reinforcement learning in this baseline algorithm for the prediction of weights. Finally, we also explored two more approaches that focused on directly predicting the movement of a drone.
In the evaluation we observed that the baseline algorithm and the RL model focusing on the principles had the most reliable results.
We also saw many limitations of the other RL models. One problem is the decision on the reward function and other conditions like the truncated condition. If they are not chosen wisely, the model will not reach the desired goals. Also, many more learning iterations may be needed in order to achieve reliable prediction results for the movement vector.
Additionally, it might not be necessary or even always possible in a real-world environment to change the movement vector each frame.
There is a vast number of options to adjust these models and explore them further. First, even more training iterations with a more diverse set of environments may help the models to improve the prediction results. Additionally, instead of predicting the movement vector, it might be beneficial to predict a change in the yaw and pitch angle of a drone, imitating the movement of the leader drone and limiting the option of the model.
Additionally, it would be interesting to test the models with a larger number of drones in the simulation. However, this may also increase the update time significantly, as we already saw higher computation times for the executions with many surviving drones.

References

[Ada20] Land Adams. Boids. Website, 2020. Accessed: October 29, 2024, https://people.ece.cornell.edu/land/courses/ece4760/labs/s2021/Boids/Boids.html.

[AHR21] Rina Azoulay, Yoram Haddad, and Shulamit Reches. Machine learning methods for uav flocks management-a survey. *IEEE Access*, 9:139146–139175, 2021.

[Kim23] Jonghoek Kim. Leader-based flocking of multiple swarm robots in underwater environments. *Sensors*, 23(11), 2023.

[Lag19] Sebastian Lague. Coding adventure: Boids. YouTube video, August 2019. Accessed: September 26, 2024, https://www.youtube.com/watch?v=bqtqltqcQhw.

[SWL09] Housheng Su, Xiaofan Wang, and Zongli Lin. Flocking of multi-agents with a virtual leader. *IEEE Transactions on Automatic Control*, 54(2):293–307, 2009.

[VVT+13] Csaba Virágh, Gábor Vásárhelyi, Norbert Tarcai, Tamás Szörényi, Gergő Somorjai, Tamás Nepusz, and Tamás Vicsek. Flocking algorithm for autonomous flying robots. *Bioinspiration and Biomimetics*, 9, 10 2013.