Nico Winkelsträter

Evolutional Configuration Optimization of Autonomous Quadcopters

FAKULTÄT FÜR
INFORMATIK

Intelligent Cooperative Systems
Computational Intelligence

Evolutional Configuration Optimization of Autonomous Quadcopters

Master Thesis

Nico Winkelsträter

October 25, 2022

Supervisor: Prof. Sanaz Mostaghim

Advisor: Dr. Christoph Steup

Nico Winkelsträter: *Evolutional Configuration Optimization of Autonomous Quadcopters*
Otto-von-Guericke Universität
Intelligent Cooperative Systems
Computational Intelligence
Magdeburg, 2022.

Contents

List of Figures

List of Tables

1. Introduction

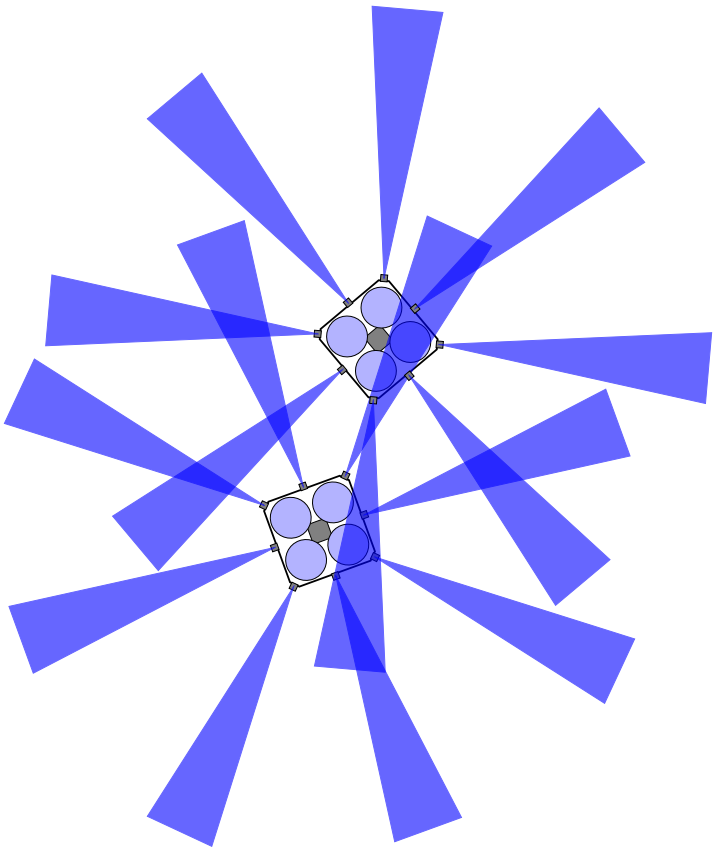


Figure 1.1.: 2D Illustration of the copter models with sensors used in the simulation

1.1. Motivation

Unmanned aerial vehicles (UAVs), including unmanned quadcopters, are very flexible and are used in many applications. They are used in agriculture, surveillance, forest fire monitoring, building inspection [18, 16, 23, 25] etc.

Many of these applications could benefit from using a swarm of UAVs by covering a lot of area more quickly, providing better signal-to-noise ratio by using distributed sensing [24, p. 11] or higher fault tolerance. If it is clear in advance that a particular task will deploy a swarm of UAVs, it is beneficial to try to adapt and optimize the UAVs behavior and sensors to better suit the task or save costs. Doing this can be a difficult task, and is also known as the *design problem* [26, p. 48]. This work explores the feasibility of using evolutionary algorithms to overcome the design problem for swarms of quadcopters.

1.2. Research Questions

The following research questions are proposed in this work:

RQ1 Is it possible to co-optimize structural and behavioral parameters of a swarm of quadcopters performing an aggregation behavior using an evolutionary algorithm?

RQ2 Is it possible to perform this optimization in a fully automatic fashion?

RQ3 Is it possible to detect any impact of the structural parameters on the behavioral parameters or vice versa?

RQ4 Can this method produce solutions which outperform solutions which are manually designed by an expert?

1.3. Structure of this work

At first, Chapter 2 explains the background of the methods and techniques used in the work. Chapter 3 presents some related works and the state of the art. In Chapter 4 the working of Sensor Placement using Co-Evolution (SPOC) and all of its parameters are explained. Chapter 5 then explains the technical details of how SPOC is implemented. The experiments are explained, and their results presented, in Chapter 6. Finally, the answer to the research question is summarized and possible improvements of SPOC and future work is discussed in Chapter 7.

2. Background

This chapter describes the three fundamental topics this thesis is based on: Robot Simulation Framework, Swarm Intelligence and Evolutionary Algorithms.

2.1. Terminology

This work deals with swarms as well as evolutional algorithms. As both topics might use the term *individual* in different ways, it is important to differentiate between the two. In Swarm Robotics or Swarm Intelligence, an individual is a distinct member of the swarm. In evolutional algorithms, an individual is the encoding or instantiation of a solution to the problem that is being optimized.

All swarms described in this work are of mostly homogeneous nature — except the leader — and all members of the swarm are "assembled" identically as instructed by the solution to be evaluated. From the viewpoint of evolutionary algorithms, one could speak of the whole swarm as an individual, as it represents one solution to the problem. Therefore, the term *individual* in this work refers to a solution of an evolutional algorithm, while the parts of the swarm will be referred to as *robots* or *copters*.

2.2. Simulation

This section explains the choice of Simulation Framework, how it works and how it is configured for this thesis.

To facilitate the evaluation of individuals in the evolutionary algorithm, a simulation framework is required because physical evaluation is infeasible because of high cost and time requirements. The simulation consists of two

components: an autopilot and a physics simulator. Ardupilot[1] is used as the autopilot and Gazebo[2] as the physics simulator.

Ardupilot is an open source autopilot, which is capable of piloting many types of vehicles, including multi- and quad-copters. It is usually deployed on embedded hardware installed in the vehicles themselves. This piece of embedded hardware is typically also called autopilot. Ardupilot also supports running on typical computers as Software In The Loop (SITL) for testing and simulation purposes, and supports multiple physics simulators for providing flight data. It is extensible with custom flight modes, which is made use of in this work to implement the swarm behavior used for evaluation.

Gazebo is an open source physics and robotics simulator and is also one of the supported simulation backends for Ardupilot SITL. It uses the Open Dynamics Engine (ODE) as physics solver and therefore offers rigid body physics and collision detection. With a plugin, Gazebo can simulate the lift and drag created by the rotor blades Together with modelling the torque applied to the copter by the changing velocity of the rotor blades, this is sufficient to represent the mode of movement of real quadcopters. Differential thrust of the motors can control the yaw, pitch, and roll axis of the copter either by a difference in lift (pitch and roll) or by a difference in torque (yaw). Additionally, Gazebo offers different sensors, including a distance sensor with a conical field of view which can be used to approximate the sonar distance sensors used on the FINken. With these components, Gazebo offers everything that is required to model the FINken quadcopter dynamics and sensors. Gazebo can also be extended using plugins, which is useful to gain more control over the simulation than is usually available or add, e.g. additional sensors.

2.2.1. Simulation of the FINken

To get accurate results from the simulation, the 3D model supplied to Gazebo has to have a proper mass distribution. To achieve this, the real FINken copter was approximated by a few simple shapes that were deemed most important to the mass distribution. These are the battery, the center frame section, the outer frame section, the rotors, and the sonar sensors. The battery is modeled separately from the center section, as it represents a significant portion of

[1] `https://ardupilot.org`
[2] `https://gazebosim.org`

the mass of the whole copter and sits below the center of mass of the center structure of the copter. Modeling them both together as a single unit would shift the center of mass significantly. Table 2.1 lists the measured masses of these components and Figure 2.1 shows how they are assembled to approximate the real FINken copter. For simplicity, the sonar sensors are arranged in a circle rather than attached to the outer frame. This might have a small effect on the accuracy of the mass distribution, but as the position error is small and the mass of the sensors is small compared to the remaining mass of the copter, the impact on flight behavior is negligible.

Table 2.1.: Masses of the individual modeled copter components

Component	Mass
Center Frame	168g
Outer Frame	190g
Battery	54g
Sonar Sensor	6g
Rotor	10g

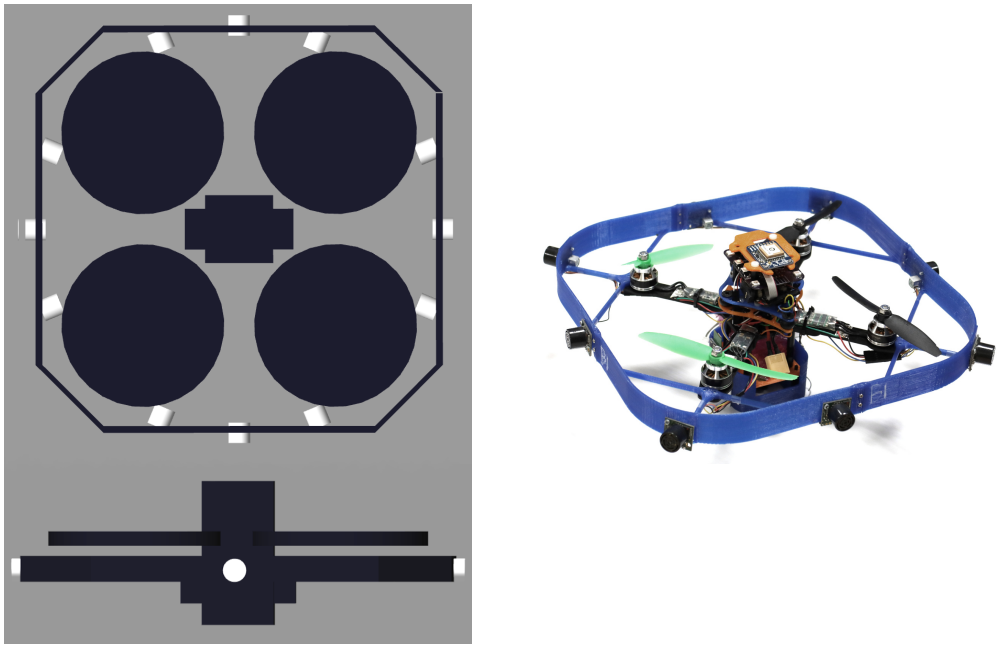


Figure 2.1.: Top and side view of the copter model and a picture of the FINKen3

2.2.2. Sensor characteristics

Gazebo provides the ability to simulate various sensor types. Gazebos sonar senors can either use a sphere or a cone as detection pattern. To approximate the sensors used on the FINken a cone with a varying opening angles and maximum ranges are used. An illustration of these sensors (not to scale) can be seen on the title page of Figure 1.1.

2.2.3. Lift Simulation

The four disks visible in Figure 2.1 represent the rotor blades and are fixed in place with the freedom to rotate. When Gazebo receives motor commands from Ardupilot it will apply a torque to these disks, and they will start to rotate. Because these disks have proper mass and inertia properties set, this will also cause a torque to be applied to the whole copter. To cancel the torque, two disks spin clockwise the other two spin counterclockwise, as is common in quadcopters. Applying differential thrust to these two sets of rotors gives the ability to control the yaw of the vehicle. The disks are important for correct torque simulation, but do not generate any lift on their own. To generate lift, each disk has two lifting surfaces defined, representing the two blades of a rotor. Gazebo will apply forces to the lifting surfaces depending on the air speed and angle of attack. The lifting surfaces are rigidly attached to the rotor disks so that the lift and drag forces are properly applied to the copter.

2.2.4. Copter Guidance

The swarm behavior which will later be introduced relies on the *Guided* flight mode of Ardupilot. In this mode, the copter does not respond to inputs from a human with a remote but to commands from a ground station, a piece of software running on a computer sending flight commands and receiving telemetry data. The intended purpose of this mode is to let a copter fly pre-planned missions following a set of waypoints or interactively supplying a position to fly to[3]. This mode also has other sub-modes which do not specify a target position but a target velocity or acceleration. To apply the attraction-repulsion forces to the copters, they are transformed into accelerations by dividing by the copter mass and then passed to the acceleration sub-mode.

[3] `https://ardupilot.org/copter/docs/ac2_guidedmode.html`

In order for this mode to work the autopilot needs to know the position of the copter which is usually provided by a GPS module. In this case, the position data is provided by the simulation, which has perfect knowledge of all flight data.

2.3. Swarm Intelligence and Swarm Robotics

This section describes the fields *Swarm Intelligence* and *Swarm Robotics* and how the concepts of each are applied in this thesis.

2.3.1. Swarm Intelligence

The term *Swarm Intelligence* was first introduced by Beni and Wang [4] in 1993 in their research on Cellular Robotic Systems, which is "a simplified model of general distributed robotic systems" [4]. This model assumes that the robots are not centrally controlled, have no shared memory, and can only communicate with other robots when they are in proximity. Beni and Wang proceed to define *Swarm Intelligence* as follows:

> Systems of non-intelligent robots exhibiting collectively intelligent behavior, evident in the ability to unpredictably (#) produce specific (=not in a statistical sense) ordered patterns of matter in the external environment.
>
> # Unpredictable is meant as globally 'intractable' or 'externally not-representable'.

When Beni and Wang talk about the unpredictability of the outcome of such a system, they mean — what is today known as — *emergence*. [17] A system shows *emergent* behavior when the systems' behavior cannot be predicted by the behavior of a single individual of the swarm.

The term *Swarm Intelligence* later lost its strict robotics connection and was subsequently used in the context of optimization methods and societal studies employing concepts from natural swarms [5]. A new term for swarm intelligence in the context of robotics was later introduced: *Swarm robotics* As the usage and meaning of the term *Swarm Intelligence* widened over time Beni offered a new definition in [3]:

> The intuitive notion of "swarm intelligence" is that of a "swarm" of agents (biological or artificial) which, without central control, collectively (and only collectively) carry out (unknowingly, and in a somewhat-random way) tasks normally requiring some form of "intelligence".

2.3.2. Swarm Robotics and Evolutionary Robotics

Swarm Robotics is a field of research regarding multi-robot systems which takes inspiration from swarming animals and insects to jointly achieve a goal.

Şahin and Spears provide the following definition of swarm robotics:

> Swarm robotics is the study of how numerous relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment. [24]

Swarms of animals or insects often have properties which are very desirable for robotic systems: They are robust, flexible, and scalable. Swarm robotics tries to take advantage of these properties.

A **robust** swarm system can continue operation despite failure of some parts of the swarm. There are multiple properties of insect swarms that lead to their robustness. In homogeneous swarms, each swarm member can be replaced by any other swarm member or in quasi-homogeneous swarms like ant colonies, there are likely enough other swarm members of the same role present to compensate the failure. Insect swarms also do not have any centralized entity in charge of coordination, and therefore there is no single point of failure. Instead, swarm members communicate in a local scope, which makes the whole swarm more robust to failures. Additionally, every member of the swarm is relatively simple, making them less likely to fail in the first place when compared to a monolithic complex system which could achieve the same task as the whole swarm [24].

Due to the simple and non-specialized swarm members, swarms are also **flexible**. Multiple swarm members can perform a variety of tasks with a variety of requirements by cooperation. For example, a particularly heavy object can be moved by multiple swarm members low signal strength can be compensated for by using multi-hop communication [14].

Because of the limitation of communication to a local scope, swarms, and swarm robotic systems are **scalable**: Adding or removing swarm members is possible without altering the function of the swarm because only few swarm members are affected by such a change [14].

The field of robotics, and for that matter, Swarm Robotics, has one fundamental problem to solve: the *design problem*. That is defining "appropriate individual rules that will lead to a certain global pattern" [26]. Designing the appropriate structures and rules which lead to the desired behavior requires the decomposition of the global behavior into the actions and interactions on the individual level. Such a decomposition might be very difficult to perform. Evolutionary Robotics is another field of research inspired by the principles of nature that tries to bypass the design problem.

> Evolutionary robotics is an automatic technique for generating solutions for a particular robotic task, based on artificial evolution. [26]

It applies evolutionary algorithms to robots or robot swarms to find a solution that shows the desired "global pattern". Often these algorithms co-evolve the morphology (body) and the control (brain) — as it also happens in nature — because it results in fitter individuals [1].

2.4. Evolutionary Algorithms

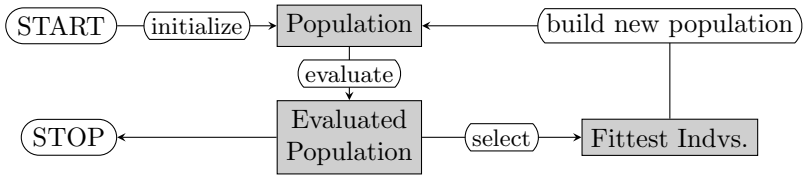


Figure 2.2.: Flowchart of a regular Evolutionary Algorithm (adapted from [26])

Evolutionary algorithms are a class of metaheuristics which employ concepts inspired by biological evolution [17]. Metaheuristics are used to find approximate solutions for which no exact or efficient, i.e., non-exponential, solutions are known. This is usually done by performing a guided random search, which iterates on candidate solutions with some measure of solution quality for guidance. To produce new solutions which are similar to existing, good

solutions, two mechanisms are employed by evolutionary algorithms: *mutation* and *crossover*. Mutation is the process of randomly altering the genes of an individual, and crossover is the process of combining the genes of parent individuals in to new individuals. In most cases, random mutation is not beneficial or even harmful to the fitness of an individual. But some mutations might lead to a small increase in performance/fitness. When combined with a selection mechanism that only lets the fittest individuals reproduce and many repetitions, this leads to a gradual increase in fitness. A typical structure of an evolutionary algorithm is shown in Figure 2.2.

According to Kruse et al. [17, p. 237] the following building blocks are required for an evolutionary algorithm.

- an **encoding** for the solution candidates
- a method to create an **initial population**
- a **fitness function** to evaluate the individuals
- a **selection method** on the basis of the fitness function
- a set of **genetic operators** to modify chromosomes
- a **termination criterion** for the search
- values for various **parameters**

An encoding is a way to represent an individual in a way which makes it easy to operate on it with the genetic operators. This is usually a list of values, each describing a property of the individual, but this might vary significantly depending on what works best for the problem the algorithm is applied to. At the beginning of the algorithm the encodings of the initial population need to be initialized with a random initialization suggesting itself as the obvious method. Although, often there are constraints applied to the values that are produced by the initialization and genetic operators as some encodings might not produce a valid individual or are deemed infeasible.

2.4.1. Multi-objective Evolutionary Optimization

To evolve a robots' form or behavior, the desired outcome needs to be measurably defined. But there is rarely a single metric which describes perfectly the desired behavior and therefore "it is common to find in the literature fitness functions composed of multiple behavioral terms" [27], which basically expresses a multi-objective problem as a single-objective problem. This leaves

the problem of weighting the different terms against each other to find a suitable trade-off up to the designer, which is sometimes very difficult because the different terms might even conflict with each other. Instead, the objectives can be separated by searching for *Pareto-optimal*/non-dominated solutions.

Definition 1 (Strict Dominance, from [17])
An element $s_1 \in \Omega$ strictly dominates an element $s_2 \in \Omega$ if s_1 dominates s_2 and there is an i with $1 \leq i \leq k$ such that

$$f_i(s_1) > f_i(s_2)$$

Definition 2 (Dominance, from [17])
An element $s_1 \in \Omega$ dominates an element $s_2 \in \Omega$ if and only if for all $1 \leq i \leq k$

$$f_i(s_1) \geq f_i(s_2)$$

If a solution is not strictly dominated by any other solution, it is Pareto-optimal. Searching for Pareto-optimal solutions removes the need for a single compound fitness function and instead allows the user to choose the importance of the objectives a posteriori. However, in the setting of an evolutionary algorithm this makes it more difficult to use the typical selection operators as a simple comparison now does not work anymore. The solutions are spread on the *Pareto frontier*, which is the set of Pareto-optimal solutions. A multi-objective evolutionary algorithm should select solutions evenly from the Pareto frontier to offer a diverse set of good solutions. The most popular [17] approach to achieve this, is the Non-Dominated Sorting Genetic Algorithm-II (NSGA-II) [8]. It uses the dominance criterion to rank the solutions and also includes a mechanism to remove solutions which lie too close to each other on the Pareto frontier to ensure a diverse set of solutions.

3. State of the Art and Related Work

Due to the prohibitively high cost of performing evolutionary optimization techniques on actual hardware, most of the current research resorts to using simulation, although some research makes used of the increased availability of rapid prototyping as well as a modular design to reduce these costs [1].

Multi-objective optimizations (MOO) has only been applied sparsely in evolutionary robotics, but Trianni and Lopez-Ibanez[27] show that it produces good results and solves some common problems of single-objective optimization, like the bootstrap problem and premature convergence. According to [27] MOO also allows a wider exploration of the objective space and the use of proxy objectives when no objective exists that explicitly defines the desired outcome. They consider some common problems used in Swarm Robotics like Maze Navigation, flocking and a strictly collaborative task and solve them using equivalent multi- and single-objective methods. They conclude that MOO can be especially beneficial in evolutionary robotics when no "good fitness function is available a priori".

Table 3.1 shows a selection of works which in some way try to optimize the "body" or "mind" of a robot for a certain task and are categorized by a couple of properties. The works differ in the applied method, used robot, goal, or task the robot has to achieve, which parts of the robot are changed and what type of controller software is used.

The *evolution* column of the table shows which parts of the robots are changed with evolution. This is either body or mind, or both. *Body* refers to the physical construction of the robot. This is further divided into sensors and structure, as some works focus on the sensor properties while others focus on the moving components that make up the robot. *Mind* refers to the control software of the robot, the part that takes data from the sensors or other environmental

data and controls the actuators of the robot. The mind might be a genetic program, a neural network, a set of stimulus response rules or a state machine.

The *method* column shows which optimization method was used. This mostly includes single-objective evolutionary algorithms (EA), multi-objective evolutionary algorithms (MO-EA) and multi-objective genetic programming (MO-GP). Notably, not many works chose a MOO method. The *swarm* column simply indicates whether the method was applied to a single robot or to multiple robots engaging in a swarm behavior, and the *task* shows what type of behavior the robot or swarm was optimized for. The *robot* column shows the name or type of robot used, and the *controller* column shows what type of control software was used.

Co-evolution of body and mind As already previously noted and found by [1] many works evolve both body and mind, likely because it generally leads to fitter individuals. Co-evolution of both of those is common because usually changing the construction of a robot also necessitates a change in the controller in order to respond to the new construction correctly. For example, in [7] the shape of a fin on a fish-like robot and oscillation rate are co-evolved to optimize for speed and efficiency in a multi-objective fashion and different fin shapes might require the robot to oscillate in a different frequency to achieve the best possible fitness for this shape.

Goals and Task A very common optimization goal is Travel or Locomotion, which means the robots are optimized to move as far, fast or efficiently as possible. This goal is mostly applied to single robots [13, 6, 20, 15, 19, 21] while for swarms the most common goal is Flocking/Aggregation or Pursuit [9, 11, 28].

Types of robots When categorizing robots into nautical, terrestrial and airborne types the most common type in evolutionary robotics seems to be terrestrial robots with two thirds of the reference papers from Table 3.1 choosing wheeled or legged robots. This is likely because experiments on these types of robots are easier to perform than on nautical or airborne vehicles, and because especially the wheeled robots can be approximated very closely with simple kinematic models for simulation.

Controller Most works require a custom controller software specifically designed for the hardware of one specific robot. This work stands out as it re-uses a common autopilot software with only a few additions so that it could theoretically be used on many different muticopter robots.

	Evolution		Method	Swarm	Task	Robot	Controller
	Body	Mind					
Haller [13]	Str.	x	EA	—	Travel	Neubot	C (Neuron Oscillators)
Parker [22]	Sen.	x	EA	—	Foraging	Hexapod	C (Stimulus Response Rules)
Bugajska [6]	Sen.	x	EA	—	Travel	UAV	C (Stimulus Response Rules)
Moore [20]	Str.	x	EA	—	Travel	Amphib.	C (Motor Oscillation)
Hornby [15]	Str.	x	EA (L-Systems)	—	Travel	"Turtle"	C (Motor Sequence)
Clark [7]	Str.	x	MO-EA	—	Loc.	Fish	C (Preset Motor Sequence)
Barlow [2]	—	x	MO-GP	—	Pursuit	Plane	C (Genetic Program)
Dorigo [9]	—	x	EA	x	Agg.	s-bots	C (Perceptron)
Francesca [11]	—	x	F-Race	x	For./Agg.	e-puck	C (State Machine)
Yasuda [28]	—	x	EANN	x	Pursuit	Two-wheel	C (Neural Network)
Mohid [19]	—	x	EIM	—	Travel	Pi-Swarm	C (Carbon Nanotubes)
Mwaura [21]	—	x	GEP	—	Travel	Cellular	C (Genetic Program)
This work	Sen.	x	SPOC	x	Agg.	FINKen	Ardupilot (Attraction, Repulsion)

Abbreviation	Meaning
Str.	Structure
Sen.	Sensor
C	Custom Controller Software

Table 3.1.: Comparison of other works using evolutionary optimization techniques on robots

4. Sensor Placement Optimization Using Co-Evolution (SPOC)

4.1. Behavior

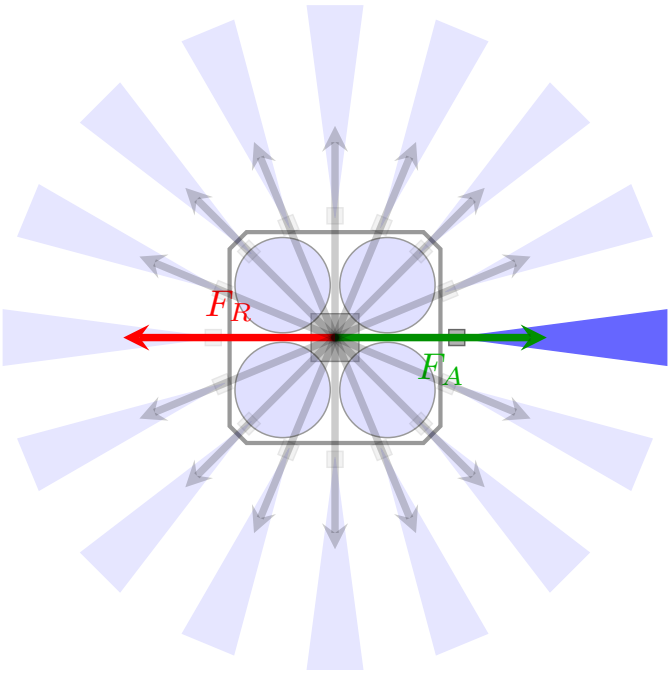


Figure 4.1.: Attraction and repulsion forces generated by the sensors

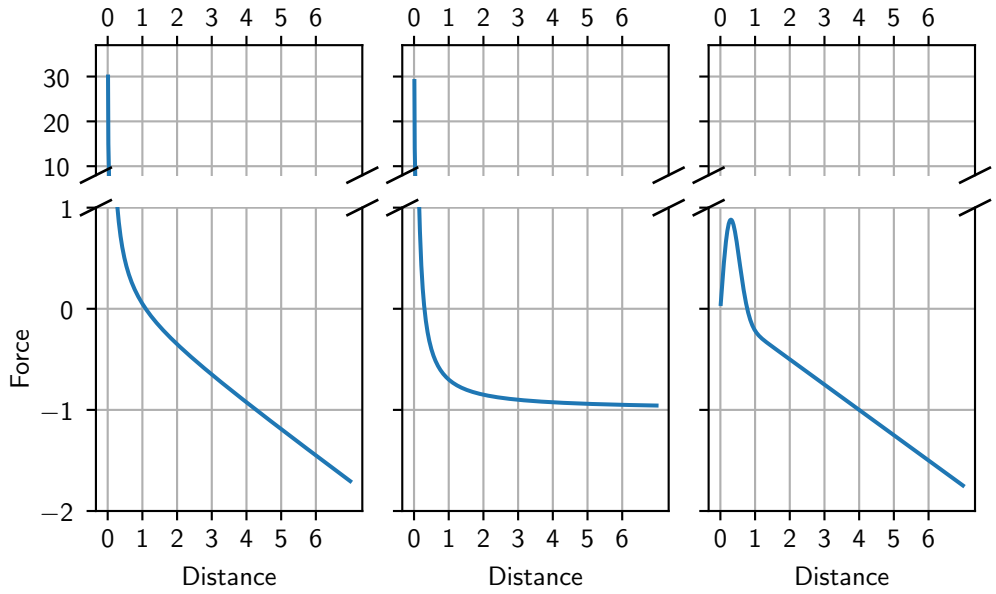


Figure 4.2.: Examples of the attraction repulsion functions

Algorithm 1: Swarm behavior

Data: Y =List of sensor yaws relative to copter
l =Flag: Is this copter the leader?
Input: S =List of sensor readings
y =yaw of the copter relative to world
Result: Force F

$\vec{F} \leftarrow l \cdot \vec{F}_{leader}$;
for $i \in$ `range(len(`S`))` **do**
  if *not* `detected`(S_i) **then**
    continue;
  end
  $\overrightarrow{dir} \leftarrow (\sin(Y_i - y), \cos(Y_i - y), 0)$;
  $F_i \leftarrow$ `attract_repulse(`S_i`)`;
  $\vec{F} \leftarrow \vec{F} + F_i \cdot \overrightarrow{dir}$;
  $i \leftarrow i + 1$;
end

For purposes of evaluation, a simple swarm behavior based on attraction and repulsion is implemented as shown in Algorithm 1. Each sensor measurement

is translated into a force using an Attraction-Repulsion Function. If there is no other copter within the range of the sensor, i.e., it reports its maximum range as measurement, it is not used for final force calculation. Additionally, to the forces generated by the sensor readings, the leading copter gets the constant force $\vec{F}_{leader}$ added to its resulting force $\vec{F}$. A commonly used attraction-repulsion function used for swarms is the one presented by Gazi and Passino[12]. A swarm with agents using this attraction-repulsion function is proven to have a fixed swarm center, and any free agent will move towards the swarm center once it has passed a certain distance from the center. These properties are proven under the assumption that there are no communication delays and require infinite detection range. In this application, though, there are delays introduced by sensor processing done by Ardupilot and the sensor detection range is finite by design. The sensors are the only means of communication between the robots, and in order to fit the intuition of swarm intelligence, the communication should be limited to a local scope. As neither of these conditions are met in this application, it is possible that this attraction repulsion function cannot result in a stable swarm. In order to account for this possibility, the experiments will explore a few different functions as well.

In the following listing the function output is only the magnitude force, the direction is determined in the behavior code by the yaw angle of the corresponding sensor. Attraction repulsion function III is the one presented by Gazi and Passino.

Attraction Repulsion Function I

$$F^I(r) = r \cdot (P_a - P_b/r^2) \tag{4.1}$$

Attraction Repulsion Function II

$$F^{II}(r) = r \cdot (P_a/r - P_b/r^2) \tag{4.2}$$

Attraction Repulsion Function III

$$F^{III}(r) = r \cdot (P_a - P_b \exp(-r^2/P_c)) \tag{4.3}$$

An example of each function can be seen in Figure 4.2. Function I and II require two parameters, while Function III requires three parameters. But the number of parameters for function III can be reduced to two by defining a comfortable

distance so that all functions require the same number of parameters. The comfortable distance is the point where the attraction force F_A is equal to the repulsion force F_R.

$$F_A = \vec{p}\, dP_a = \vec{p}\, dP_b \exp\left(\frac{-d^2}{P_c}\right) = F_R \tag{4.4}$$

$$\implies P_a = P_b \exp\left(\frac{-d^2}{P_c}\right) \tag{4.5}$$

$$\iff \ln P_a = \ln P_b \frac{P_c}{d^2} \tag{4.6}$$

$$\iff d = \mp\sqrt{P_c \ln \frac{P_b}{P_a}} \tag{4.7}$$

From Equation (4.6) follows that with a preset comfortable distance d and given P_a and P_b, $P_c = d^2 \ln \frac{P_b}{P_a}$

4.2. Objectives

The desired swarm behavior is a uniform aggregation of swarm members, moving in a mostly straight line. To achieve this behavior, two objectives are defined: aggregation quality and motion quality, which are both inspired by the objectives used by Dorigo et al. [9]. Objectives are calculated from the last 25 data points, so the last 2.5 seconds of the simulation. It is assumed that the behavior of the swarm will take some time to converge because the copters might initially not be at the distances the attraction repulsion desire and also the copters will take some time to accelerate. This is why only data from the last few seconds where the swarm had enough time to converge to a stable behavior is considered. An effort is made to make the execution of all simulations as equal as possible as is explained in Section 5.1.1 but there might still be a difference of a few data points, this is an additional reason to select the same amount of data from each copter instead of all the available data.

4.2.1. Aggregation Quality

Dorigo et al. [9] use the optimal solution to the circle packing problem as a measure for how well the robots aggregated themselves. For a given swarm size and robot diameter, there exists the smallest circle of radius r_m into which all the robots can be fit. If a robot is closer than r_m to the swarm center it achieves an aggregation quality 1, if it is further away the aggregation quality is linearly reduced until it reaches some threshold k further than r_m, at which point the aggregation quality is set to 0. There are two problems with using this exact measure of aggregation quality for this work. Firstly, the copters are not supposed to touch each other unlike the robots from [9] and secondly it requires finding a suitable value for k and no method for finding it is given by Dorigo et al. [9]. In order to avoid having to find a well working virtual size for the copters to be able to use this measure while allowing distance between the copters and having to find a value for k a measure based on the distribution of the distances of the copters to the swarm center is used (see Figure 4.3). The distances of all copters is collected and the variance, i.e., the mean of the squared differences from the average center distance, is used as a measure of aggregation quality. This means that no longer the most compact formation is preferred, but formations in which the copters have mostly equal distances to the center of the swarm. This allows different sizes of formations with maybe different sensor ranges and attraction repulsion parameters to occur while still showing a uniform aggregation pattern.

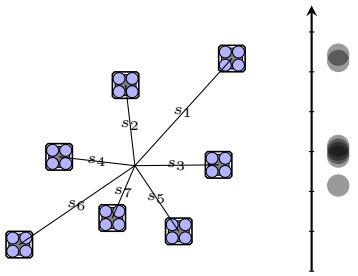


Figure 4.3.:
Example of the distances used for calculation of aggregation quality.

4.2.2. Motion Quality

Motion quality is a measure of how straight the motion of a copter was. It is adapted from the motion quality used by Dorigo et al. [9] for two-wheeled robots, where it is calculated by summing the speed differential of the two wheels over the course of the experiment. To transfer the idea of straight-line motion to flying vehicles, the sum of the angles between three subsequent positions of the vehicle is used, as shown in Figure 4.4. The motion quality of

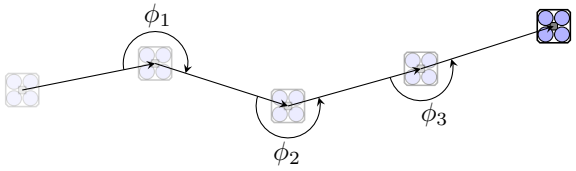


Figure 4.4.: Example of the angles used for computing motion quality

a copter c is $Q_M(c) = \sum_{i=2}^{n} \texttt{angle}(p^c_{i-2}, p^c_{i-1}, p^c_i)$ where `angle` is the smaller of the two angles between the three points supplied. The total motion quality of an individual is the average motion quality of all the copters. As the objective calculations always use the same number (25) of data points, there exists a maximum value for the motion quality of 23π. This maximum value is used to scale the motion quality to a range from 0 to 1. The final motion quality for an individual is shown in Equation (4.8).

$$Q_M = \frac{\sum Q_M(c)}{|C| \cdot 23\pi}, c \in C \tag{4.8}$$

4.2.3. Certainty

Due to the nature of the physics-based simulation in use, the evaluation is not deterministic and the fitness value of an individual will vary between successive evaluations. If not accounted for, this might lead to the accidental selection or dismissal of individuals which, by chance, achieved an above or below-average result. To avoid this problem, two mitigations have been applied to the fitness function.

The first mitigation is to apply a rolling average over each objective of an individual that has been evaluated multiple times in an unchanged form (see Figure 4.5). This means that the older the individual is — i.e., the more often it has been evaluated — the more confident one can be of the fitness value of it. But this alone will not resolve the problem, the solutions with higher confidence need to be preferred in some way by the selection operator in order for this to mitigate the problem.

Young individuals could be penalized by multiplying the fitness with a sigmoid-like function in order to prefer older individuals with higher certainty of their fitness. But this adds more complexity and the need to figure out what type of penalty function works best. Instead, the age of an individual is added as an

additional objective that shall be maximized. This way older individuals will be preferred automatically without the need to find a good penalty function while still allowing good, young individuals to be selected.

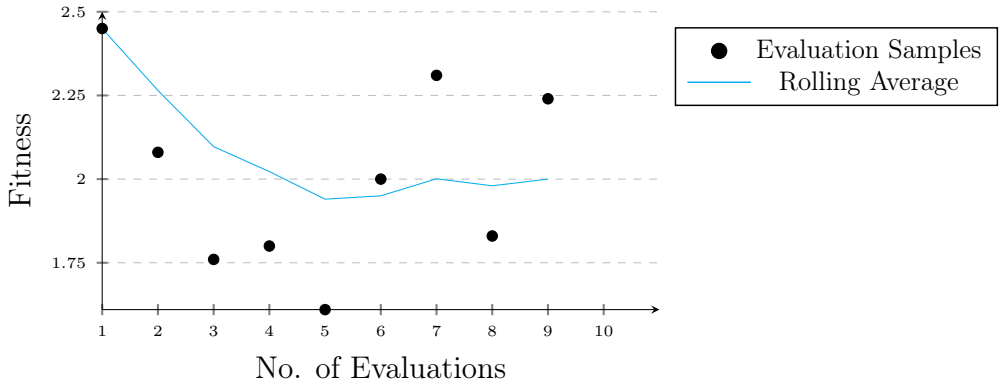


Figure 4.5.: Example of fitness scaling with multiple evaluations

4.3. Algorithm

This section describes in detail the evolutionary algorithm used for solving this multi-objective problem. First, the general structure is shown and then all the components are explained individually.

The pseudocode is listed in Algorithm 2 and visualized as flow-chart in Figure 4.6. At first, the archive and population is initialized. The archive is initialized to the empty set and the population to a list of N random individuals. A random individual is assigned 16 sensors, each with a randomized yaw, opening angle and maximum range. Additionally, the parameters for the attraction repulsion function are added, chosen from a uniform distribution. The Archive is a set in which every individual created during the runtime will be saved. An individual consists of the genome as well as fitness history and a computed fitness.

`evaluate` Each individual is then evaluated by the process described in Section 6.1 and are the inserted into the archive.

`update` When the archive is updated with the new population, any new individual is directly inserted into the archive. If the individual has been evaluated before, the new fitness value is added to its history and a new effective fitness

Algorithm 2: Copter Evolution

Data: N =Population Size
M =Number of Individuals to select from archive
C =Co-evolution period
Result: *pop*: List of individuals

```
1  archive ← ∅;
2  pop ← N random Individuals;
3  gen ← 0;
4  while gen < MAX_GENS do
5      pop ← evaluate(pop);
6      archive ← update(archive, pop);
7      parents ← selNSGA-II(M, archive);
8      offspring ← parents;
9      for o1, o2 ∈ loop_pairs(offspring) do
10         if gen mod 2C < C then
11             o1, o2 ← crossover_sensor(o1, o2);
12             o1 ← mutate_sensor(o1), o2 ← mutate(o2);
13         else
14             o1, o2 ← crossover_ar(o1, o2);
15             o1 ← mutate_ar(o1), o2 ← mutate(o2);
16         end
17         offspring ← fill_to_n(offspring, N, o1, o2);
18         if |offspring| = N then
19             break;
20         end
21     end
22     gen ← gen + 1;
23 end
```

value is calculated like described in Section 4.2.3. The archive used in SPOC behaves differently from how archives are typically used in evolutionary algorithms, it does not have a fixed size and includes all individuals instead of only non-dominated ones. Because of the non-deterministic nature of the simulation and the resulting variance of the fitness, it is desirable to keep even non-dominated solutions — especially regarding the certainty objective — in the archive to give them a chance of getting re-evaluated and gain certainty of their fitness.

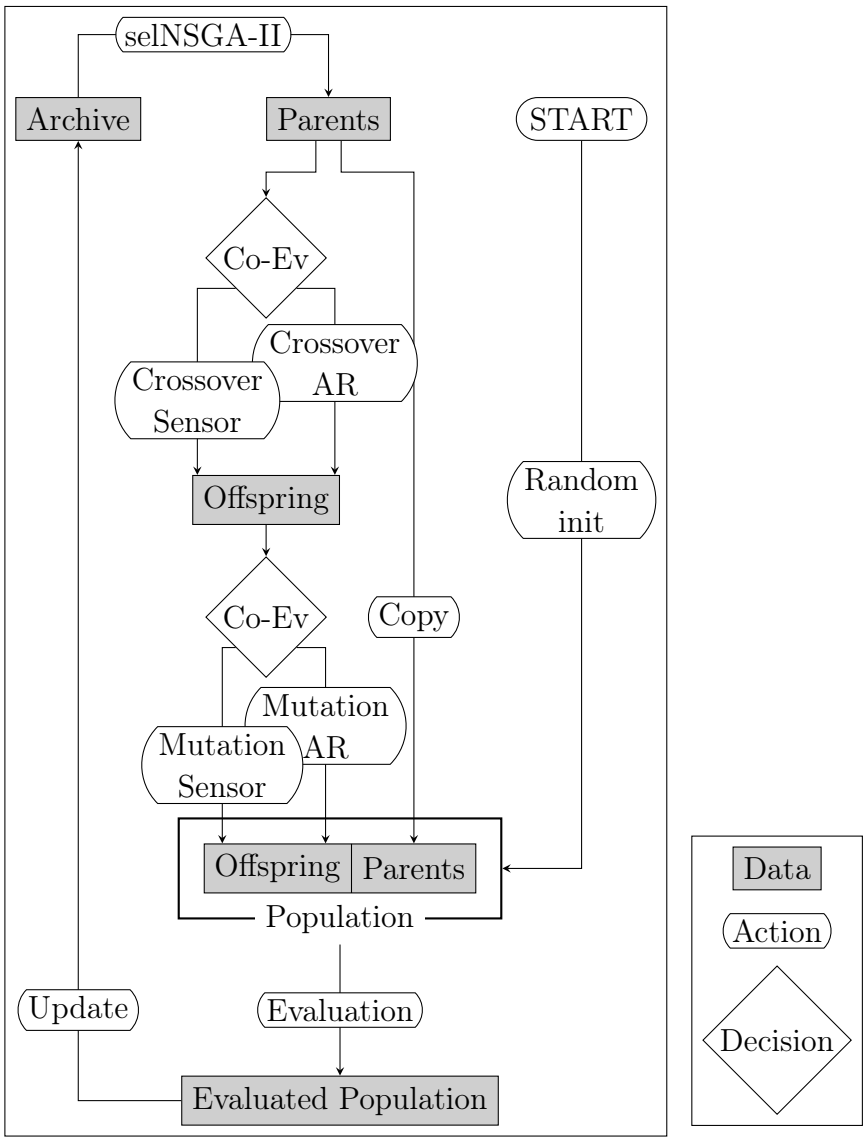


Figure 4.6.: Flowchart of SPOC

`selNSGA-II` Using the selection Operator from NSGA-II [8] M individuals are selected from the archive to be the parents of the next generation.

The parents are grouped into pairs and from each pair two offspring are generated using a two-point crossover, and both offspring might be mutated. Depending on which co-evolution cycle is currently active, either only the sensors or just the attraction-repulsion parameters are changed during crossover and mutation.

`fill_to_n` New offspring are generated from the pairing of the parents until the new population reaches a size of N. If $N - M$ is an odd number at the end of the filling loop, only one of $o1$ and $o2$ can be added. `fill_to_n` is used to discard $o2$ if necessary. The parents are included in the new population so that they can be evaluated again in order to improve the confidence in their fitness values.

4.3.1. Encoding

There are two parts to be encoded: The sensor position, orientation and properties, and the attraction-repulsion parameters.

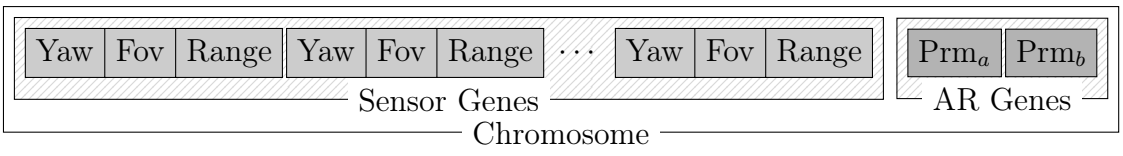


Figure 4.7.: Encoding of sensors and attraction repulsion parameters

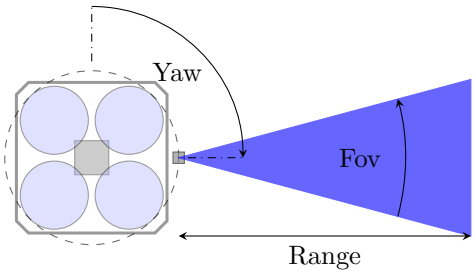
Attraction-repulsion parameters The attraction-repulsion functions used require either two (ARI, ARII) or three (ARIII) parameters. One of the parameters for the third function ARIII is redundant and can be determined by the other two if a comfortable distance is given. With this property, only two parameters are needed for any attraction repulsion function because all sensor forces use the same parameters.

The initial range for initialization of the attraction repulsion parameters is one of the experiment variables.

Sensor position and properties For maximum flexibility, the sensor position could be encoded as Cartesian coordinates relative to the robot center

and a pair of yaw and pitch angles. However, this would lead to a vast number of solution which are infeasible because they are either in positions so far from the robot that an actual physical structure to hold the sensor in that position would be large and heavy or because they are in position without visibility, like pointing at the robot itself or in directions not relevant for the 2D experiment.

In order to eliminate such solutions and limit the search space, the sensor positioning is restricted. Sensor positions will be limited to a circle, and the sensors will point radially outwards from the center of the copter. With these restrictions, only one angle needs to be encoded for the position of each sensor.



Parameter	Min	Max
Yaw	0°	360°
Fov	8°	40°
Range	2 m	7 m

Figure 4.8.: Top-down view of copter showing the parameters of a sensor and a table showing the possible value ranges of those parameters.

The possible sensor properties are based on the same line of products used on the FINken so that it should be possible to find actual sensor which are close to the sensor parameters produced by the evolutional algorithm.

The final encoding consists of two floating-point values for the attraction repulsion parameters and $3n$ floating-point values for the sensor yaw, field of view and maximum range, see Figure 4.7.

4.3.2. Mutation and Crossover

The mutation operator is based on a simple Gaussian mutation, with a standard deviation σ determined by the range of possible values of the considered gene (Equation (4.9)). Values for field of view and maximum range are clipped to the valid range if the mutation causes it to leave the range. Attraction-repulsion parameters as well as the yaw of the sensors do not have any bounds and no clipping needs to be done. If the yaw of a sensor leaves the range of [0,

2π] it is disabled, allowing the algorithm to produce solutions with different numbers of sensors. A Gaussian mutation was chosen over a random mutation to allow the gradual improvement of sensors. Bounded polynomial mutation is another possible mutation operator, but it was not chosen because of a mix of bounded and unbounded genes, so with a Gaussian mutation with optional clipping the same mutation can be used for all genes.

$$\sigma = p \cdot (g_{max} - g_{min}) \tag{4.9}$$

p is the *mutation scaling* factor to scale the standard deviation σ, depending on the valid range of the corresponding parameter. Valid ranges for the sensor parameters are shown in Figure 4.8. The parameters for the attraction-repulsion functions are not limited except for the constraint that they are non-negative. Therefore, the range from the initialization distribution is used, which ranges either from 0 to 10 or from 0 to 25. Figure 4.9 shows this with an example parameter which has an initial range from 0 to 10 with $p = 0.1$, $p = 0.25$ and $p = 0.5$ resulting in $\sigma = 1$, $\sigma = 2.5$ and $\sigma = 5$.

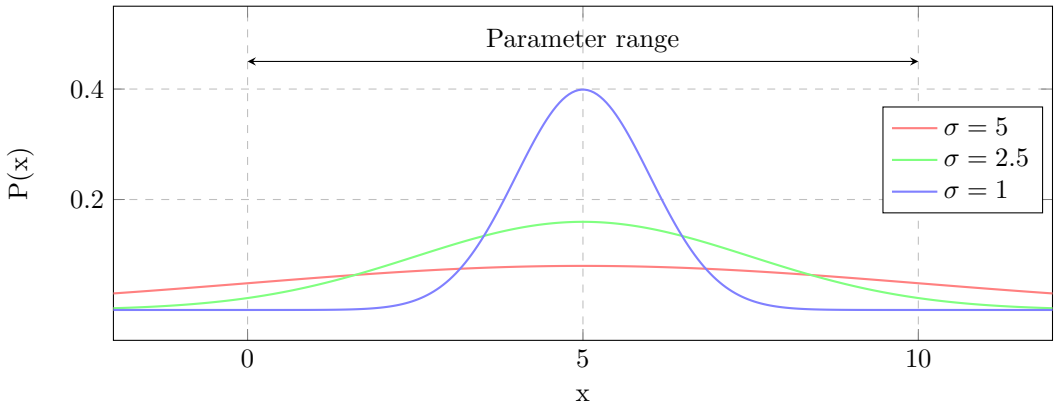


Figure 4.9.: Normal distributions with different standard deviation used for Gaussian mutation of genes.

Sensor properties and attraction repulsion parameters are co-evolved in cycles. Depending on the co-evolution cycle, only the respective parts of the gene are affected by the mutation operator. There are only very few attraction repulsion genes compared to sensor genes. With a flat chance of mutation per gene, the attraction repulsion parameters would only change very rarely. In order to

avoid this problem, the mutation rate is normalized so that P_M represents the average number of individuals which have a single gene mutated. So for example, with $P_M = 0.2$ every 5th individual has a gene changed independent of how many genes are eligible for change in the current co-evolution cycle. Therefore, the actual mutation probability per gene P_m depends on the current co-evolution cycle and the number of genes considered by it.

There are two genes for attraction repulsion parameters and 48 genes for the sensors, and therefore the mutation probability for attraction repulsion genes is $P_M/2$ and the probability for sensor genes is $P_M/48$.
The attraction repulsion genes might be swapped after mutation if AR Function III is used and $P_a > P_b$. The other two AR functions do not have this requirement, and the parameters are never swapped after mutation.
For crossover the default two point crossover from the deap python library [10] is used, again only the genes corresponding to the current co-evolution cycle are affected. During crossover the sensor genes are handled as atomic units, i.e., only complete sensors are exchanged between individuals not just parts of a sensor gene like yaw, field of view (fov) or range like it is indicated by the contiguous gray boxes in Figure 4.7. Doing it this way stems from the hypothesis that not some particular sensor parameter is beneficial to the behavior, but a sensor or set of sensors as a whole. Performing crossover in this way also makes it simple to generate valid individuals from it and also keeps a diverse set of parameters instead of something like averaging of two parent genes.
For crossover of the attraction-repulsion parameters, a single point crossover is used, as there is only one point to cross over at. The second of the two parameters are swapped between the chromosomes.

4.3.3. Selection

Selection is done using a combination of the selection from the NSGA-II algorithm and an archive of all non-dominated solutions. After evaluation, the archive is updated with the results, then the new population of size n is created by selecting several individuals from the archive using the NSGA-II selection operator. The number of individuals selected this way is determined by the *re-evaluation rate* R. With a population size n, the number of selected individuals is $p = \lfloor n \cdot R \rfloor$. The other $n - p$ individuals are created by mating the first p individuals from the archive by paring them randomly.

5. Implementation

This chapter describes the technical details and challenges of the implementation. From the software that is used over what plugins and extension were created to how all the parts work and communicate together.

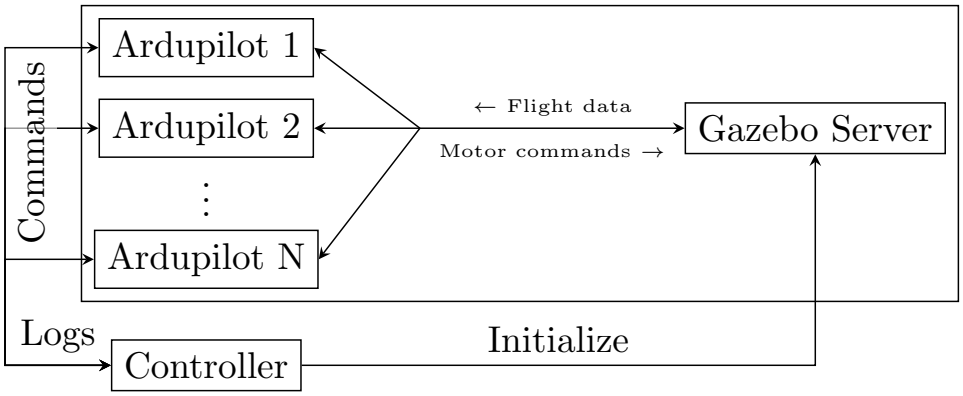


Figure 5.1.: Communication structure of simulation environment

Figure 5.1 shows a simplified overview of what processes are needed and how they communicate with each other.

Gazebo is the physics simulator that does all the lift, drag, and inertia calculations for the rotors as well as the range calculations for all the sensors. Ardupilot is the autopilot that receives telemetry and flight data from Gazebo via a tcp connection, uses this data to compute the required throttle for each rotor and sends those back as commands to Gazebo. The Controller node is responsible for managing all the Ardupilot and Gazebo processes, as well as initializing the Gazebo simulation and controlling the startup and liftoff of all the copters.

5.1. Controller

The main challenge to solve for running the evolutionary algorithm is ensuring that all experiments get a fair treatment on a heterogeneous compute cluster, as well as managing all the processes and inter-process communication needed to collect the required data.

5.1.1. Ensuring Fairness

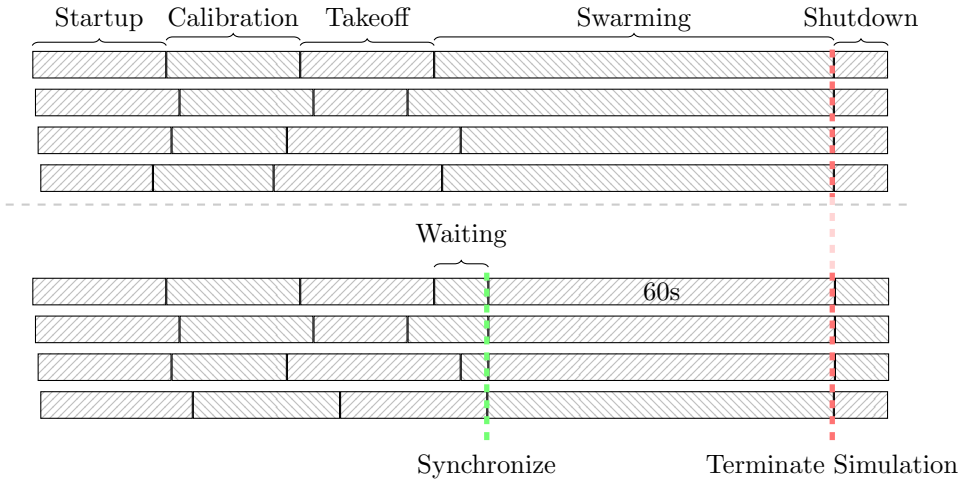


Figure 5.2.: Top: Sequence of an unsynchronized simulation run.
Bottom: Sequence of a synchronized simulation run.

The experiments are run on a heterogeneous compute cluster, and Gazebo is mostly a single threaded workload. This means that the simulation speed varies greatly depending on the performance of the hardware any particular experiment happens to be executed on. Low performance on a weaker compute node cannot be compensated by simply using more CPU cores. Additionally, the startup of all the processes, calibration procedure of Ardupilot as well as the takeoff of the copter take a varying amount of time (see Figure 5.2). This leads to two problems that need to be solved. A: The copters do not start the swarm behavior at the same time, and B: Experiments simulate different amounts of time when run for the same amount of real time.

Problem A is solved by letting the copters wait for further instructions once they reach their target altitude and become ready and signal their readiness to

the controller process. Once the controller has received the readiness notifications of all the copters, it instructs all copters simultaneously to switch into the swarm behavior. This introduces a waiting period (see bottom of Figure 5.2) so that copters with a fast initialization phase do not start too early.

Problem B is a common problem for simulations and as such Gazebo already has a feature that attempts to solve this. It allows the user to specify how many iterations of the physics solver should be performed and after those are done the simulation is stopped. However, this does not work in this case because the initialization phase does not always take the same number of iterations. If this was used, different experiment runs could spend different times in the swarm behavior phase, which has to be avoided to make the experiments comparable. To solve this, a Gazebo plugin was implemented which has the ability to communicate with the controller process. Once all the copters are ready, the controller process signals to the plugin that the swarm phase has started. As the plugin has access to the Gazebo memory, in can save how much time has been simulated up to that point and can terminate the simulation once another 60 seconds have been simulated. This ensures that in every experiment exactly 60 seconds of the swarm behavior are simulated independently of how long the startup procedure of Ardupilot took or of how slow or fast the cpu of the compute node is.

5.1.2. Process Management

For each individual to be evaluated, a Gazebo process, an Ardupilot process for each copter of the swarm and a ground station connection to interface with each Ardupilot process is required. All of these process need preparation before starting them like configuration files, parameters, or 3D model files. Ardupilot has a vast list of parameters that can be set in order to configure the vehicle that is controlled. This includes PID gains for position and attitude controllers and motor and sensor configuration. These parameters need to be properly set so that Ardupilot can properly control the copter and knows about the sensors positions and properties. It can read the parameter values from a file which needs to be created for each copter and put into the correct location so that Ardupilot reads the correct parameters for the corresponding copter. An excerpt of the parameters relevant for the swarm behavior is shown in Listing 5.1. The attraction repulsion parameters, the chosen attraction repulsion function, number of sensors and the sensor position and ranges are

passed to Ardupilot. The index of the proximity sensors starts at 2 because the first sensor is reserved for altitude measurement. Furthermore, the field of view of the sensors does not need to be passed to Ardupilot because it is not relevant for the calculation of the attraction repulsion forces.

Listing 5.1: Example of an Ardupilot parameter file

```
SWARM_AP_PRM_A 9.88
SWARM_AP_PRM_B 29.45
SWARM_AP_PRM_C 0.05
SWARM_AP_FUNC 2
SWARM_N_SENSORS 16
RNGFND2_YAW 1.4859839656955884
RNGFND2_TYPE 100
RNGFND2_MAX_CM 299
RNGFND3_YAW 4.729669710631424
RNGFND3_TYPE 100
RNGFND3_MAX_CM 519
...
RNGFNDF_YAW 0.7128277076925214
RNGFNDF_TYPE 100
RNGFNDF_MAX_CM 409
RNGFNDG_YAW 3.3937599299406154
RNGFNDG_TYPE 100
RNGFNDG_MAX_CM 347
RNGFNDH_YAW 1.0244225307885766
RNGFNDH_TYPE 100
RNGFNDH_MAX_CM 463
```

The controller is a python program responsible for managing all the processes and collection all the data needed for the experiments. The output of the evolutionary algorithm is a list of floats describing the copter configuration, which the controller converts into Gazebo compatible model files and copies and positions the models in the way that is necessary for the current experiment. Once all model files are ready, all necessary processes are spawned. Communication channels to each Ardupilot process are established using the MAVLink protocol. This channel is used for issuing commands for takeoff, transitioning flight mode, as well as recording position data of the swarm members.

5.2. Gazebo

To make Gazebo suitable for this work, it needs to have the ability to send data to and receive commands from Ardupilot as well as be able to terminate the simulation after a fixed amount of simulated time has elapsed.

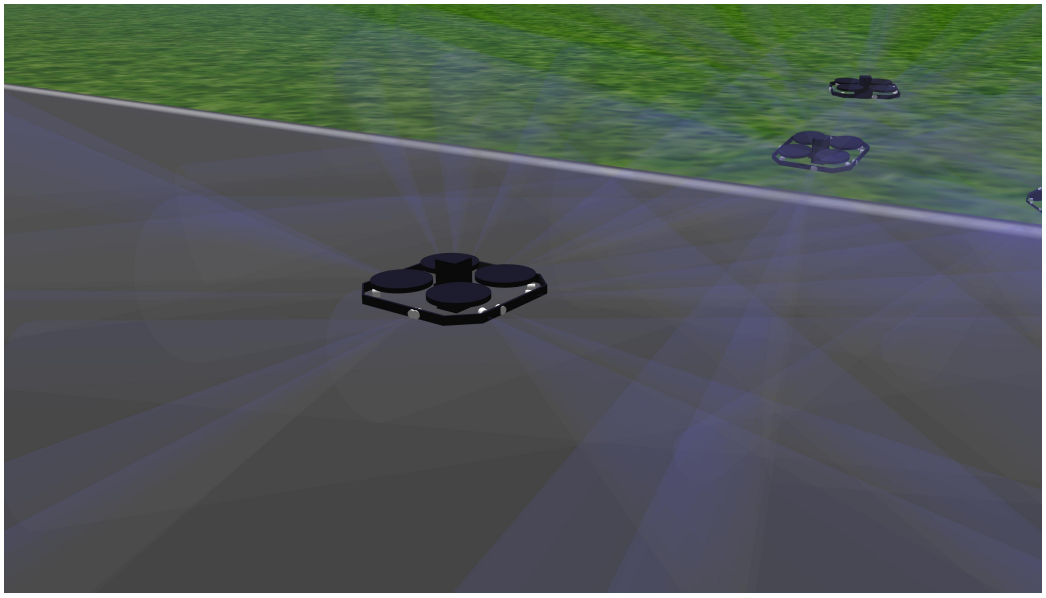


Figure 5.3.: Screenshot of Gazebo with multiple copters. The transparent cones indicate the detection area of the sensors.

5.2.1. Communication

The first requirement was partly already met by a community provided Gazebo plugin[1] and corresponding patches to Ardupilot. Using a network socket, the plugin creates a bidirectional data link between the Ardupilot process and the Gazebo process. The plugin sends flight data like velocity, acceleration, position, and orientation to Ardupilot and Ardupilot respond with the desired motor speeds. The plugin provides the ability to define rotational links in a Gazebo model, representing the rotors, to which the plugin will apply a torque in order to rotate the link at the speed demanded by Ardupilot. Ardupilot already has the ability to read sensors, including proximity sensors, and provide the data to the flight modes, but the plugin did not provide the ability

[1] `https://github.com/SwiftGust/ardupilot_gazebo`

to send sensor data from Gazebo. The plugin was therefore extended with the ability to send proximity sensor data to Ardupilot and additionally the maximum amount of proximity sensors in Ardupilot was increased to 16 as it was previously limited to 8.

5.2.2. Sensors

Section 2.2 mentioned the use of Gazebos built-in Sonar sensors and the initial implementation used these sensors. However, when running the experiment, it was discovered that these sensors do not behave as expected. The sensors use the underlying physics library to query for collisions between the cone of the sensor and any other objects to find the distance to objects within the cone. It turns out that the physics library only reports collisions with the surface of the cone and not with its volume. This means that copters completely inside a sensor cone would not be detected by that sensor at all. Changing this behavior was not feasible as it requires changes to the Gazebo source code. Instead, an entirely new detection mechanism was implemented with an approach that could be realized with a Gazebo plugin and the existing Ardupilot plugin was extended with that functionality. By not being able to use the built-in sensors of Gazebo, the ability to visualize them in the simulation was lost. It was added back by adding transparent models of cones without a collision mesh to every sensor, which can be seen in Figure 5.3.

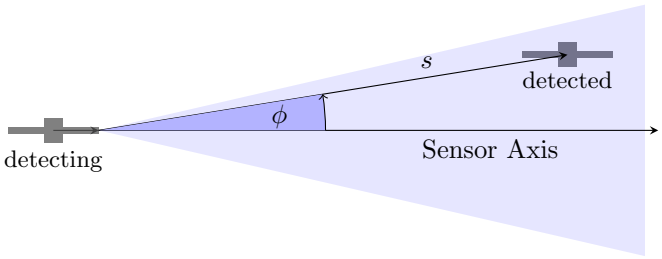


Figure 5.4.: The cone of a simulated sonar sensor in which it can detect another copter.

The new detection mechanism simply checks whether the position of a copter is within a cone for each pair of sensor and copter, like it is shown in Figure 5.4 This has the drawback that it is not a proper collision detection and copters will only be detected when the center of it has entered the detection cone, but

it is still a good approximation of the actual behavior of a sonar sensor and is significantly faster to compute than a full mesh collision.

The full detection algorithm running for each copter is listed in Algorithm 3.

Algorithm 3: Proximity Detection algorithm

Data:
S =List of sensor coordinates
$R_{\max}$ =List of Maximum detection ranges
$\phi_{\max}$ =List of sensor cone angles
Input:
C =List of copter coordinates
c =Index of detecting copter
Result: D =List of distances

```
for i ∈ range(|S|) do
    d_min ← R^i_max;
    for j ∈ range(|C|) do
        if j = c then
            continue;
        end
        s⃗ ← c_j − s_i;
        s⃗_axis ← s_i − c;
        φ ← angle_between(s⃗, s⃗_axis);
        if φ < φ^i_max and ‖s⃗‖ < d_min then
            d_min ← ‖s⃗‖;
        end
    end
    D_i ← d_min;
end
```

5.3. Ardupilot

In order to implement the swarm behavior, a new flight mode as well as a range of parameters to configure sensor parameters were added to Ardupilot.

The flight mode extends the existing guided mode of Ardupilot which already provides acceleration-based position control as a sub-mode. This makes it simple to implement the attraction-repulsion based behavior. The new flight mode simply gathers the ranging data from all sensors, ignoring the ones which

currently do not detect any other copter, calculates the combined force and divides it by the mass of the copter to get an acceleration which can be directly passed to the guided mode. The guided mode control loop then calculates the required motor thrusts to accelerate the copter by the desired amount. The SITL simulation environment is completely transparent to the code running on the copter, which means that swarm mode can theoretically run on real hardware without any changes.

6. Experiments and Results

The main experiments explore all three of the attraction repulsion functions introduced in Section 4.1 with two different sets of starting distributions and different swarm sizes.

The evolutionary algorithm has several hyperparameters which need to be set. To determine the best set of those hyperparameters, a preliminary set of experiments is performed to test different values of re-evaluation rate, mutation rate and mutation scaling factors. These parameters will then be used for all the main experiment runs.

6.1. Evaluation Scenario

This section describes the simulation procedure used for the evaluation step.

The leader is placed at the center of the simulation, all other robots are placed in a circle around the center with a radius of $2.5m$ (see Figure 6.1). The copters then begin startup in the `GUIDED` mode of Ardupilot, a mode without the sensors active, and take off to a pre-defined altitude. Once all copters report that the target altitude is reached, the controller instructs them to enter the swarm mode, in which the sensors are active and forces are applied to the copters. In this mode, a constant force is applied to the leader agent which pushes it into a pre-defined direction. This mode stays active for 60 seconds of simulated time. Ardupilot is instructed to send the position and

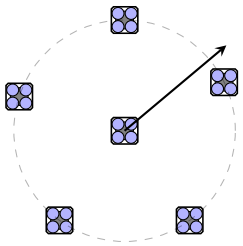


Figure 6.1.:
Initial copter formation and direction of the force applied to the leader. (not to scale)

attitude data of the copters with a rate of $10Hz$. This data is recorded and is the basis of the subsequent evaluation. Calculation of the fitness objectives is then performed using the recorded positions. Should any copters' altitude drop below 4 m during the 60 s in which the swarming mode is active, it is very likely that a collision has occurred. A penalty factor of 0.2 or 1.8, depending on whether it is minimized or maximized, is applied to all objectives for evaluation runs where this occurs.

6.2. Hyperparameter Exploration

To find a good set of hyperparameters, the preliminary experiments will use a fixed set of experimental parameters (see Table 6.1). The considered hyperparameters are population size, re-evaluation rate, mutation rate P_M, and the mutation scaling factor p of the mutation operator. All permutations of the values listed in Table 6.2 are tested, which results in a total of 54 parameter sets, all of which are listed in Table A.1. But because the evaluation is very

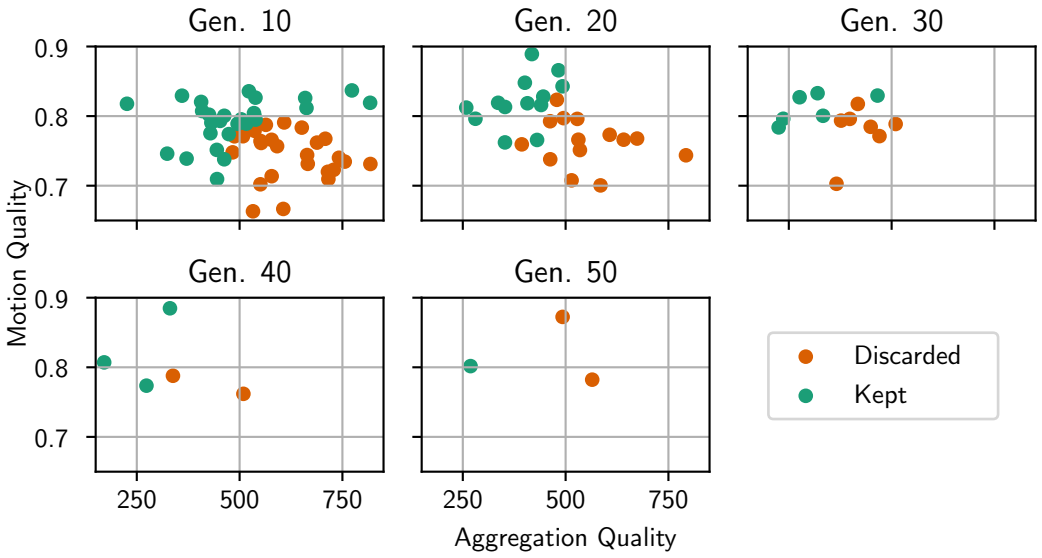


Figure 6.2.: Mean fitness values of the hyperparameter sets at each cut-off point.

costly in terms of time, not all parameter sets are evaluated for the full length of 50 generations. Instead, all parameter sets are at first evaluated for one co-

Parameter	Values
Swarm size	6
AR-Function	III
MAR Parameter range	[0, 10]

Table 6.1.: Fixed parameter values used in the preliminary experiments

evolution cycle, which is 5 generations of sensor evolution and 5 generations of attraction-repulsion parameter evolution. After that step, the half of the parameter sets which produced the worst average fitness values in generation 10 are discarded, and the other half is evaluated again for on co-evolution cycle. Selection of parameters sets is again performed with the selection operator from NSGA-II. If the amount of parameter sets is not evenly divisible, the larger part is discarded so that the following amounts of parameter sets remain after each step: 27,13,6,3,1. Using this approach reduces the total number of generations to be evaluated from 2700 to 1030 and as evaluation is costly this reduces the needed time for parameter exploration by multiple days. The population sizes 10, 16 and 22 have been chosen to try to optimally use the resources of the compute cluster used to run the experiments. The majority of workload comes from the Gazebo process, and one thread is allocated to each. An additional two threads are allocated per experiment for the controller process and Ardupilot processes, which do not require as many resources as the Gazebo processes. As such the experiments require either 12, 18 or 24 threads which lets them be nicely distributed on the compute nodes which mostly offer 48 threads. The population size of 16 was chosen as a middle ground between 10 and 22, it does not work well with the node size of 48, but the cluster has a node with 120 threads available which can be fully utilized when combing 18 thread workloads with 12 or 24 thread workloads. Figure 6.2 shows the mean fitness values of the parameter sets at each cut-off point, as well as which parameters were selected and which were discarded.

The scatter plot for generation 10 does not show an outlier with an aggregation quality of over 800 and the plot for generation 40 does not show an outlier with a motion quality of lower than 0.6 in order to make all pots more readable. Both of those outliers were discarded at the respective generation. After 40 generations, three sets remain, which happen to include one of every swarm size tested. From these three, parameter set No. 5 (Table A.1) is selected as it produced good aggregation quality and reasonable motion quality while

Parameter	Values
Population size	10, 16, 22
Re-evaluation rate	$1/3, 0.5$
Mutation rate	$0.2, 0.3, 0.4$
Mutation scaling	$0.1, 0.25, 0.5$

Table 6.2.: Hyperparameter values used in the preliminary experiments

also having the smallest population size of 10. This small population size is beneficial for the later experiments because more of them can be run in parallel when the population size is small, reducing the overall time needed for simulations.

6.3. Experiments

The main experiments explore the impact of different swarm sizes and attraction repulsion functions with different starting parameters. Due to the mostly single threaded nature of the Gazebo simulator, the swarm size is fairly limited because the time required to run a simulation increases rapidly with the number of copters in the swarm.

Parameter	Value
Swarm Size	5, 6, 7
AR-Function	I, II, III
AR Parameter range	[0, 10], [0, 25]

Table 6.3.: Experimental parameters used for the experiments

Besides the physics solver, the sonar sensor computation is one of the main contributors to the required computation time. The computation time required for the sensor calculation increases with the square of the number of copters, as each sensor has to perform the collision check with every copter. The largest swarm size was chosen to be 7. A simulation with this number of copters took roughly 45 minutes on the slowest computing nodes. With all individuals being evaluated in parallel, this results in the worst-case computation time of roughly $4\frac{1}{2}$ days per experiment. All combinations of parameters listed in

Table 6.3 are used, resulting in a total of 18 experiments, each run for 150 generations. If a single parameter set is referenced later, it will be named with a shorthand form like: ARIII 7 10, which means attraction repulsion function III, swarm size of 7 and initial attraction repulsion parameters from the range [0, 10]. Figure 6.3 shows the median fitness values of the average fitness of the experiments over time. Significant improvements in both objectives occur in both objectives within the first 20 generations. Subsequently, the motion quality is mostly stagnant while the aggregation quality is slowly improving. The age is increasing steadily but at a rate lower than one, indicating that even individuals with high age eventually get dropped from the population as better, young individuals are found. Overall, the solutions seem to have improved over time independently of experiment parameters. The reason for the anomaly in Figure 6.3 at around generation 100 is not entirely clear, but a likely explanation is that for some technical reason terminated early without being able to collect the usual 2.5 seconds of data. This would result in a low motion quality because it scales with the amount of data points. It would also likely result in a good aggregation quality because it did not have much time to move from their initial configuration, which is a circle and therefore has a low variance of the center distances.

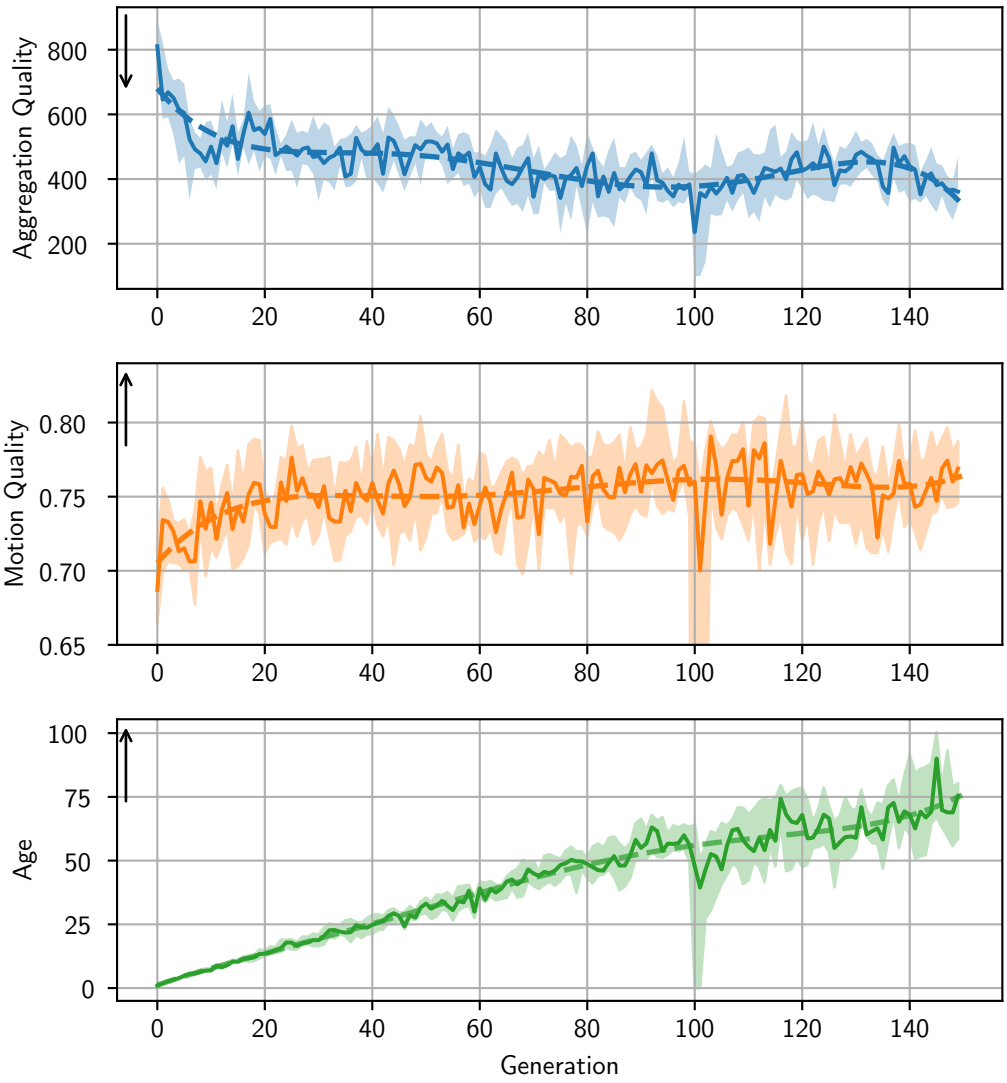


Figure 6.3.: Median values of the average fitness of each experiment with the upper and lower tercile in the shaded area. The dotted line shows a best-fit polynomial of the 5th degree.

To see which experiment parameters had an impact on solutions quality, all individuals from generation 150 are grouped together by experiment parameter and their fitness distribution plotted in Figure 6.4. Swarm size does not seem to have a big impact on motion quality, but swarm size 7 performs notably better in the aggregation quality objective. The choice of attraction repulsion function again does not seem to have a notable impact on motion quality, but

attraction repulsion function II performs notably better than the other two functions regarding aggregation quality. Initial parameter range does not have a big impact in either objective.

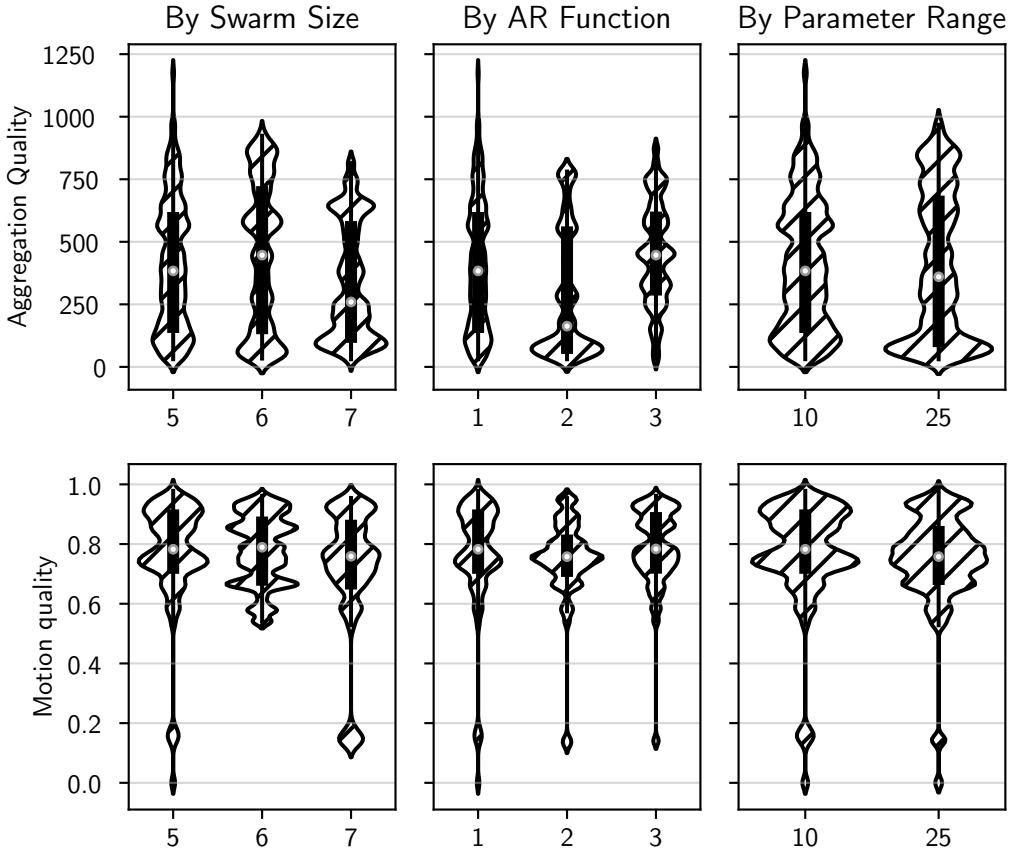


Figure 6.4.: Fitness distribution of all individuals of generation 150 grouped by experiment parameter.

When inspecting each parameter set individually in Figure 6.5, the sets ARII 6 25 and ARII 7 25 especially stand out with a good aggregation quality with a low spread, even though their motion quality does not stand out compared to other configurations. Other configurations which stand out are ARI 5 10 and ARIII 6 10 with good motion quality. The individuals produced by these configurations will be inspected more closely later.

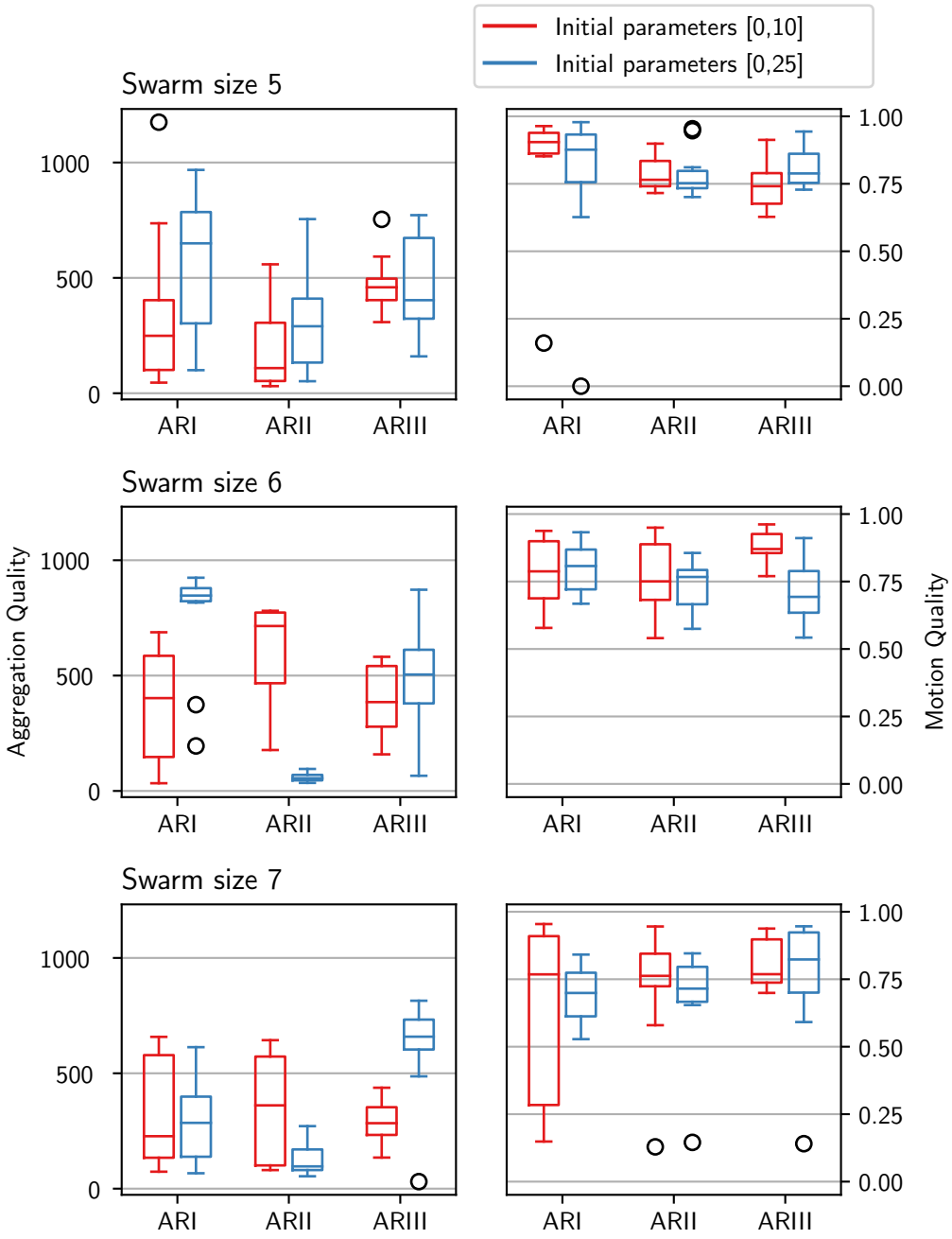


Figure 6.5.: Box plot of each experimental configuration at generation 150 grouped by swarm size, attraction repulsion function and initial parameter range for the AR function.

6.4. Comparison With Reference Individuals

To have a baseline to compare the solutions produced by the EA to, a reference copter is simulated for each attraction repulsion function and swarm size. The reference copters have 16 sensors with a maximum range of 7 m and an opening angle of 22.5° with the sensors evenly spaced around the copter similar to the configuration in Figure 4.1. The opening angle of 22.5° was chosen because with 16 sensors it grants a full 360° of coverage and the choosing the maximum of 7 m for the range of the sensors seems like the intuitive best choice for sensor performance. The attraction repulsion parameters for each function are set to $P_a = 2$ and $P_b = 1$, with a comfortable distance of $3.5m$ for AR Function III, just like in the experiments. These values were chosen because with them AR Function I and II have their zero crossing within the maximum range of the sensors, function II's bounded attraction value and the slopes of all three functions are deemed reasonable. The reference copter is evaluated 31 times for each AR function and swarm size. The resulting fitness values are shown in Figure 6.6. To compare them with the solutions produced by the evolutionary algorithm, all Pareto-optimal individuals were chosen from all individuals the EA produced combined over all generations and parameter sets. From a total of 4565 individuals, 24 were Pareto-optimal. These 24 Pareto-optimal solutions were again evaluated for 31 times each because most of them did not reach a high age during the evolution. The mean values of those runs are also shown in Figure 6.6 to compare them to the reference copter. Each large marker in Figure 6.6 marks the mean fitness of one individual, the small markers each represent a single evaluation of one individual. When observing only the means of the fitness distributions, it seems like many of the individuals selected from the results of the EA out-perform the reference individuals. That difference is most pronounced in the aggregation quality objective and is less clear in the motion quality objective.

As the fitness values have a high variance, it is not obvious by the mean values which individuals are better than others. In order to better compare the individuals, a new dominance operator is used. This operator is applied to the whole distribution of fitness values, utilizing the Mann-Whitney U test as comparison operator, using the corresponding alternative hypothesis of *greater* or *less* for maximized or minimized objectives. As acceptance threshold $p < 0.05$ is used.

Dark green markers in Figure 6.6 represent Pareto-optimal individuals from the EA which dominate all the reference individuals when using the Mann–Whitney U test for the dominance check. Most of the non-dominated solutions produced are significantly better than the hand-built reference individuals.

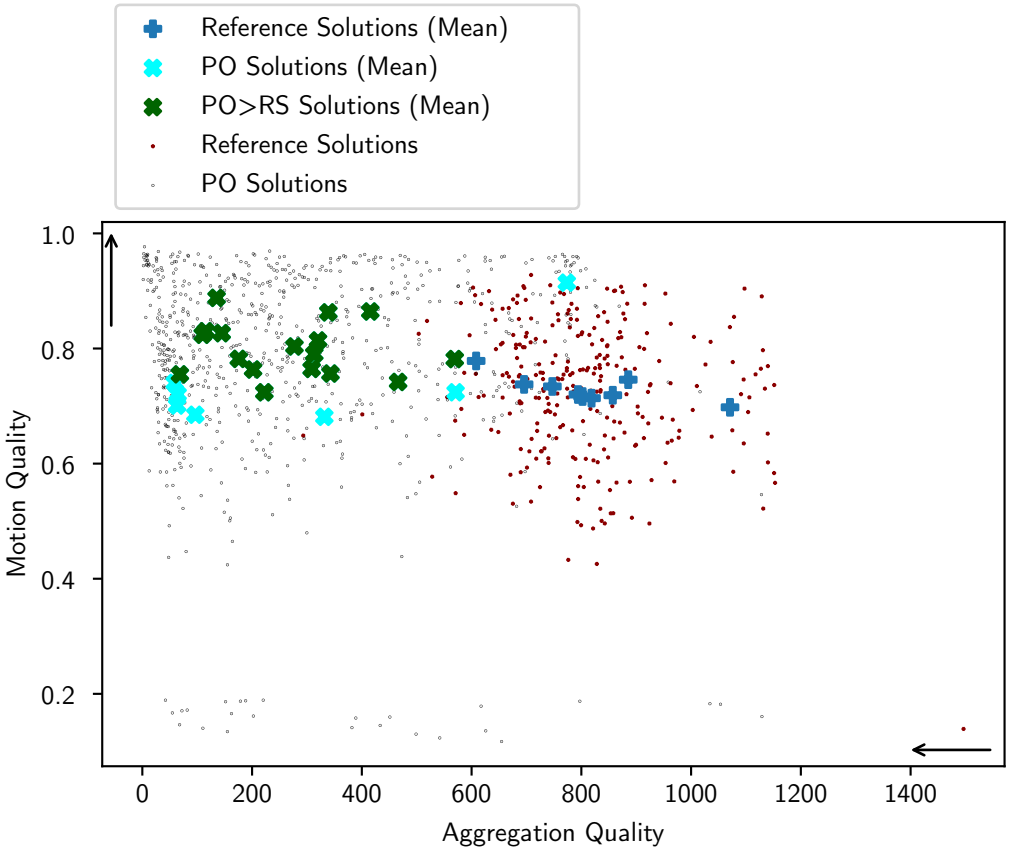


Figure 6.6.: Fitness scatter plot of all non-dominated solutions from the EA and hand-built reference solutions.

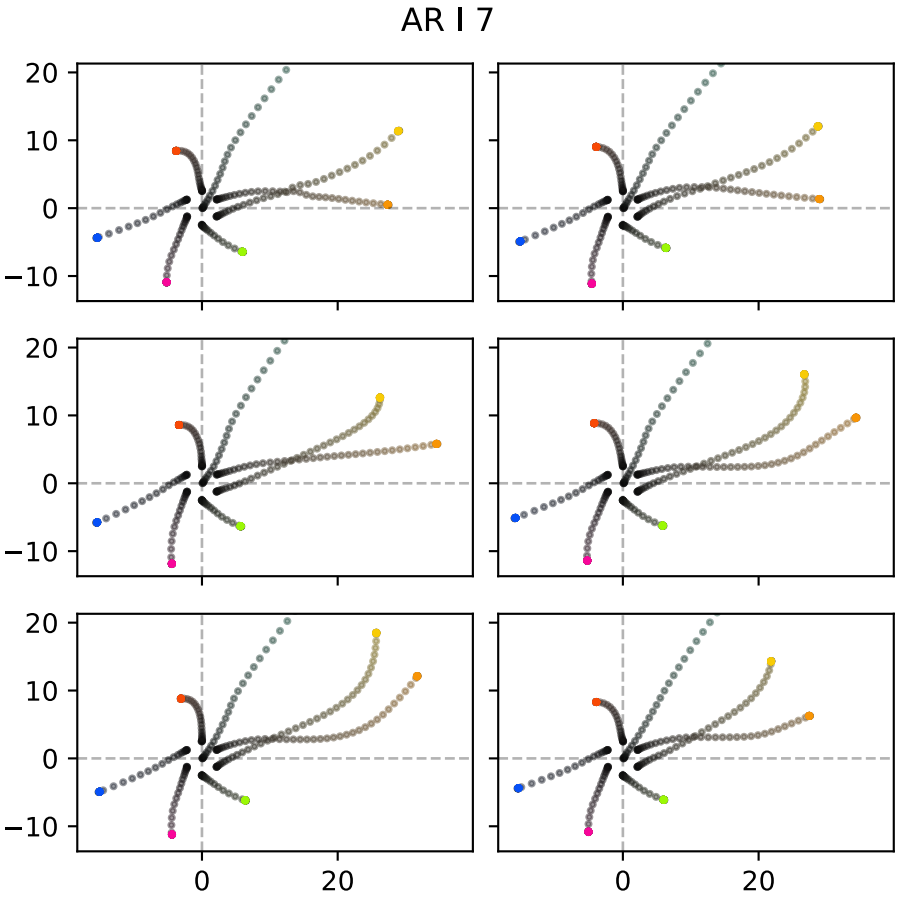


Figure 6.7.: Example flight paths of the reference individuals with swarm size 7 and attraction repulsion function I. Top-down view with scale in meters.

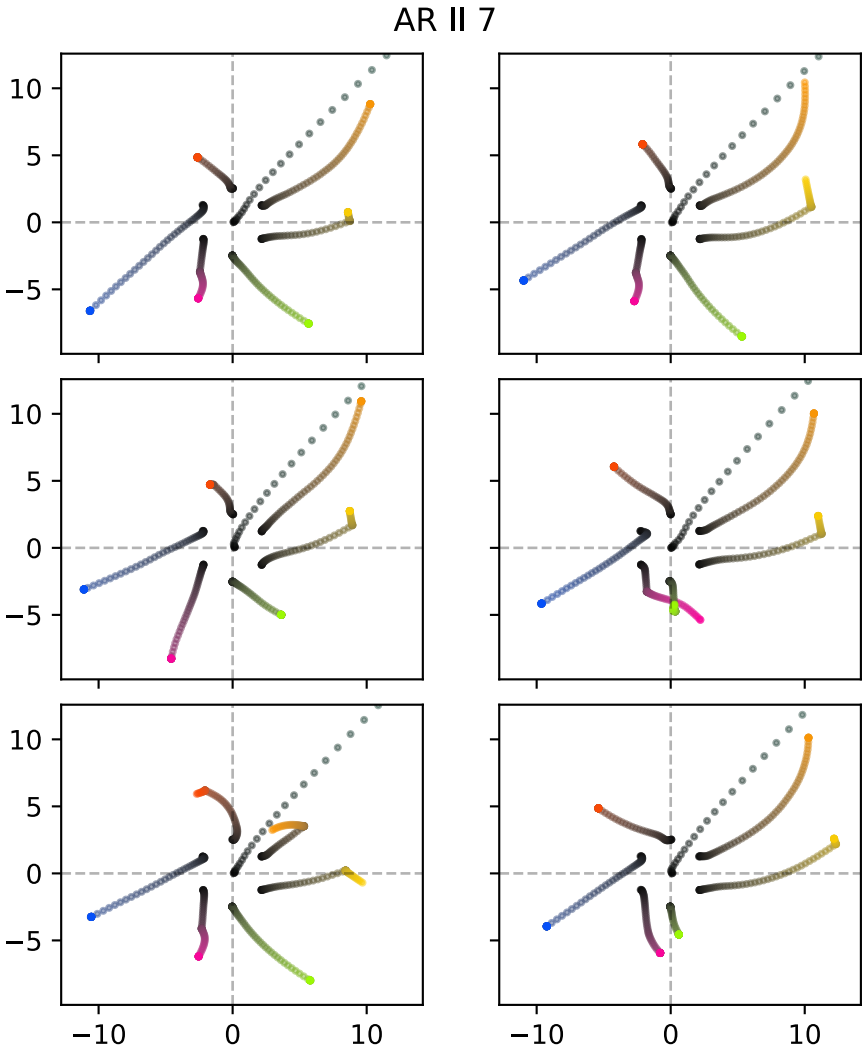


Figure 6.8.: Example flight paths of the reference individuals with swarm size 7 and attraction repulsion function II. Top-down view with scale in meters.

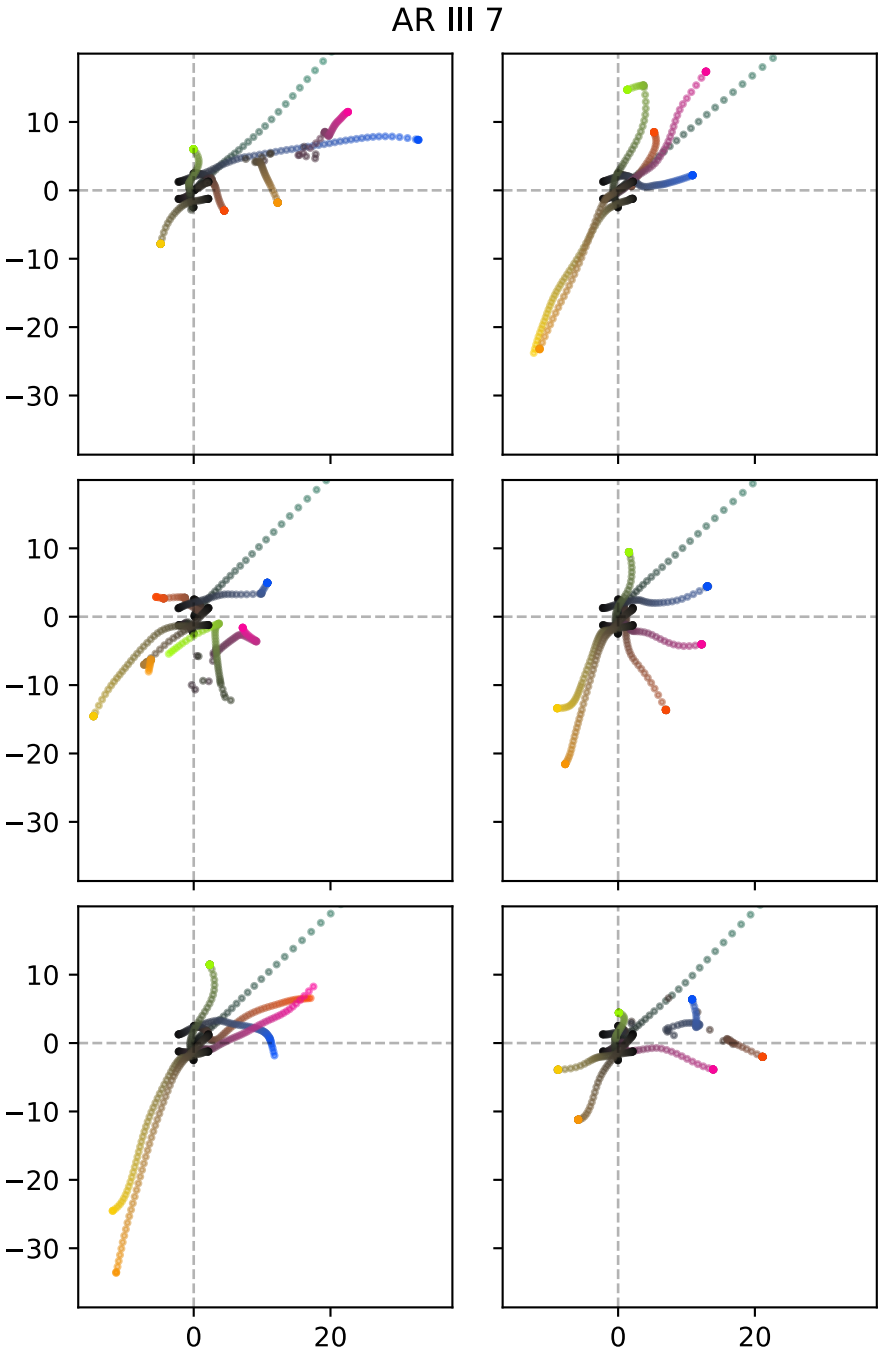


Figure 6.9.: Example flight paths of the reference individuals with swarm size 7 and attraction repulsion function III. Top-down view with scale in meters.

Figures 6.7 to 6.9 each show 6 examples of the flight paths of the reference copters for each attraction repulsion function. In all cases, the follower copters do not stay attached to the leader and the leader escapes, shown by the gray line moving out towards the top right of the graphs. If the leader escapes it usually escapes very far, and to keep the graphs compact the view was cropped to only show the paths of the follower copters. Example graphs for swarm sizes 5 and 6 are shown in Figures A.4 to A.6 and Figures A.1 to A.3. Again, the follower copters mostly cannot stay attached to the leader. With swarm sizes 6 and 7 and attraction repulsion function I and II, the copters to the right of the center seem to be noticeably affected by the leader and are pulled slightly to the top. This is most pronounced with AR function II, with a swarm size of 7 (Figure 6.8).

6.5. Inspection of Solutions

This section takes a closer look at solutions generated by parameter configurations which stood out in Section 6.3 and were also included in the Pareto-optimal set. The flight paths of some individuals, as well as the sensor configuration, is examined to understand the individual's behavior. The flight paths of the other Pareto-optimal solutions are pictured in Figure A.7..

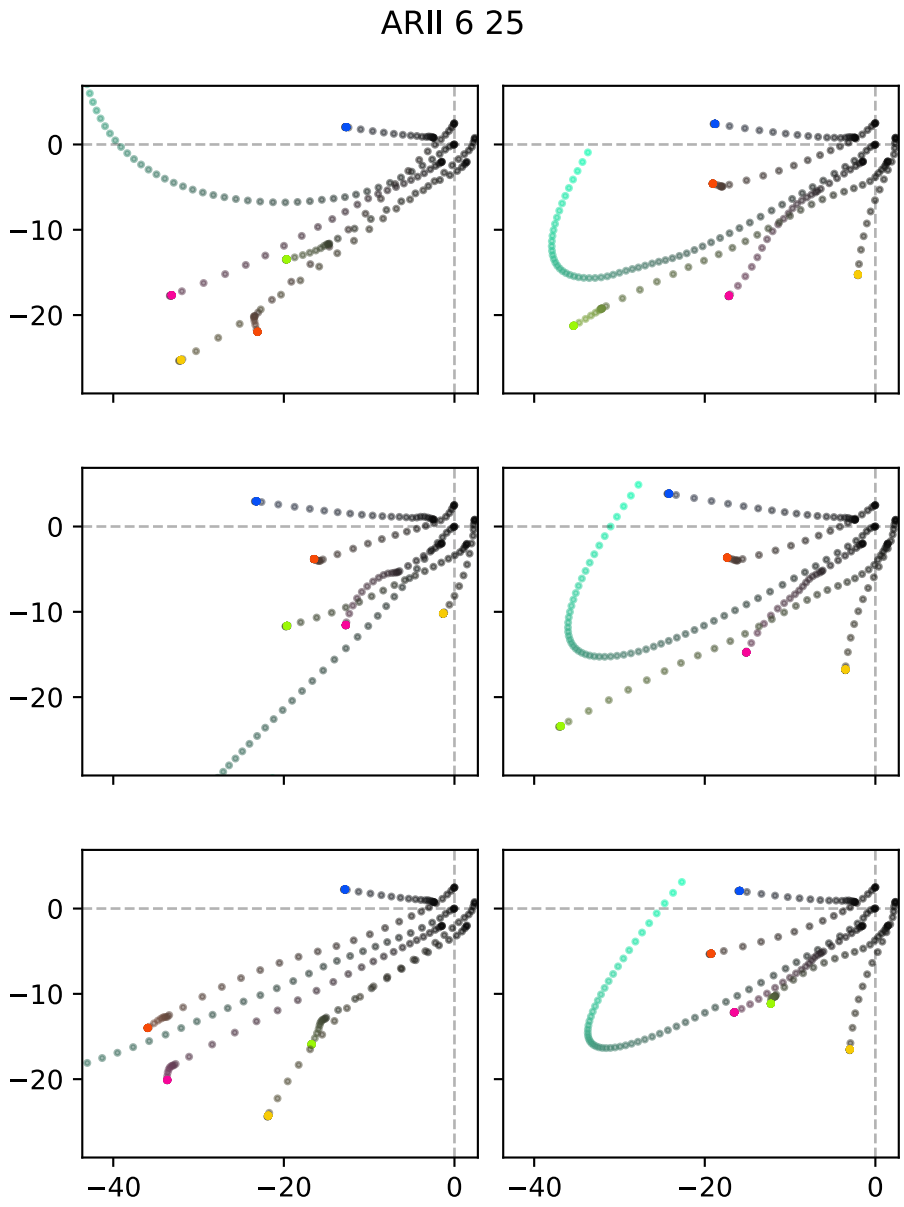


Figure 6.10.: Example flight paths of an individual from parameter set ARII 6 25

The parameter set ARII 6 25 stood out with good aggregation quality in Figure 6.5. Figure 6.10 shows some example flight paths generated by one of those individuals. The copters all move to the lower left rapidly, taking the leader copter with them, until they stop, and the leader can move clear of all

the other copters at which point its constant force takes over; it moves back in an arc to the upper right. This same or very similar behaviors are displayed by most of the individuals produced by ARII 6 25, but this one was chosen because the movement to the lower left is the most pronounced. The initial movement of the copters to the lower left happens in a very linear fashion, which explains the reasonably good motion quality these individuals achieved. But it appears that these individuals did not achieve their exceptional aggregation quality by behaving as a cohesive swarm but by spreading out in the right way, purely by chance. As the aggregation quality does not include any measure of proximity to the swarm center this is entirely possible, even if the copters spread out so far they cannot detect each other anymore and come to stop in a good position.

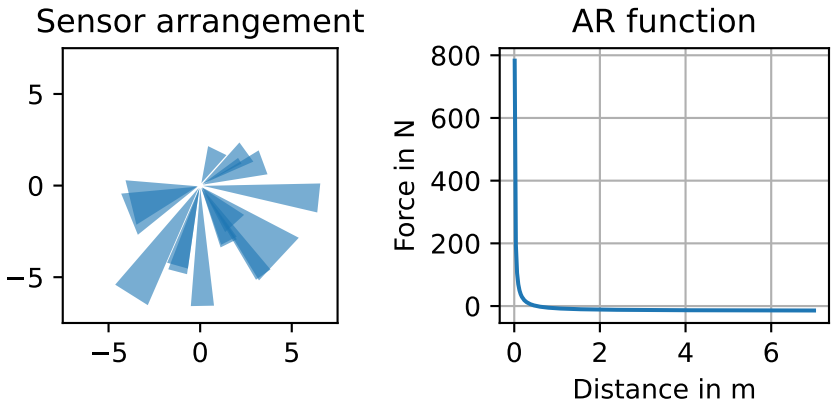


Figure 6.11.: Sensor configuration and plot of the attraction repulsion function of the individual from ARII 6 25. The blue triangles represent the area in which the sensors can detect other copters.

When looking at the sensor arrangement and the attraction repulsion function of the individual (Figure 6.11) it is not immediately obvious how this behavior emerges. The very high repulsion force probably leads to a very sudden acceleration, such that the copters loose sight of each other and therefore continue on a straight path, but the chosen direction cannot be explained without more in-depth analysis.

Another individual which stood out in the analysis of Figure 6.5 was ARI 5 10, because of its good motion quality. When inspecting some flight paths of one example individual in Figure 6.12 one can observe that this individual

can sometimes capture the leader and cohesively move all copters in the same direction. The previous individual as well as the reference individuals did not show such a behavior. There are some runs in which the leader does escape, but even in those cases at least some remaining individuals stay together and move in roughly the same direction, explaining the good motion quality.

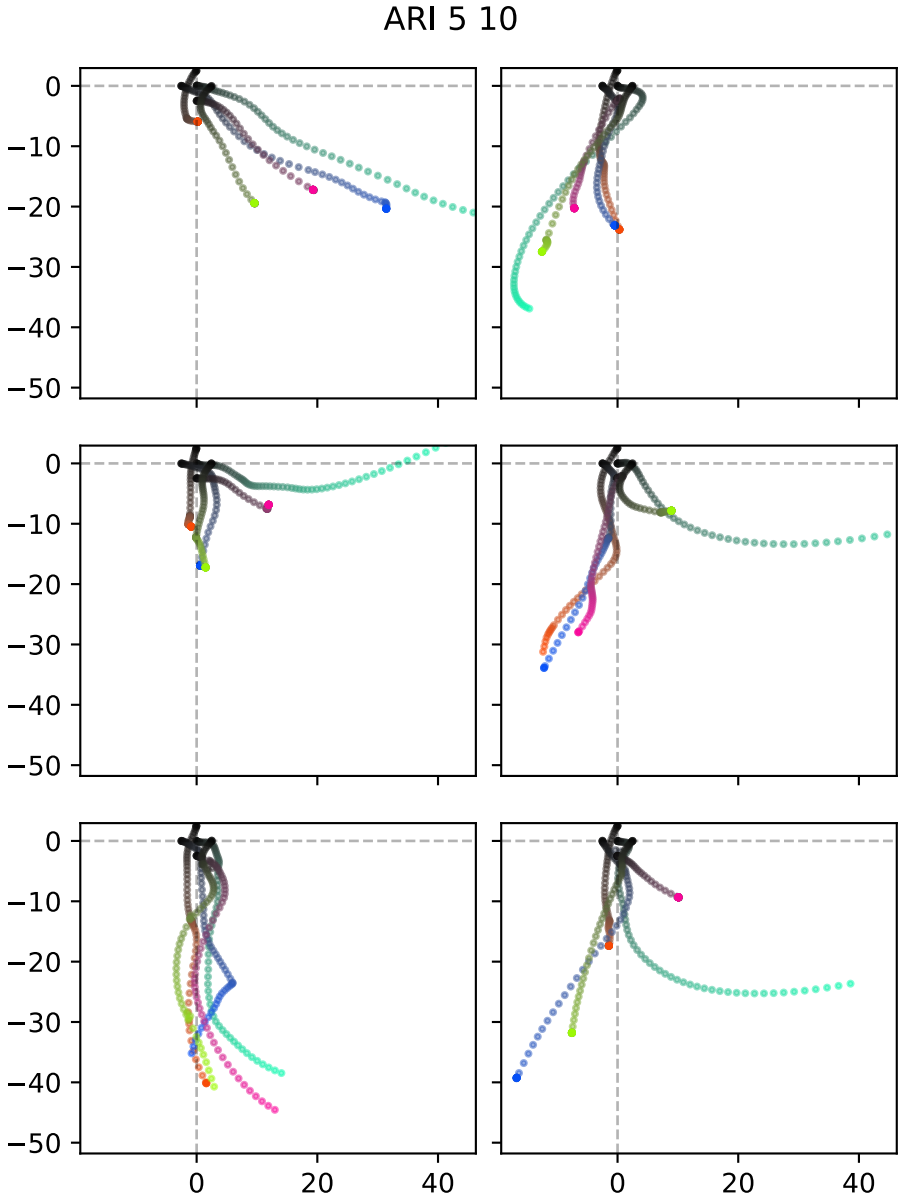


Figure 6.12.: Example flight paths of an individual from parameter set ARI 5 10

When inspecting the sensor arrangement and attraction-repulsion function (Figure 6.13), one can, again, not directly tell how this behavior emerges. But when comparing those to the previous individual one can see that the sensor have larger ranges, are more evenly spread around the copter, the repulsion forces are less aggressive and the attraction forces are higher.

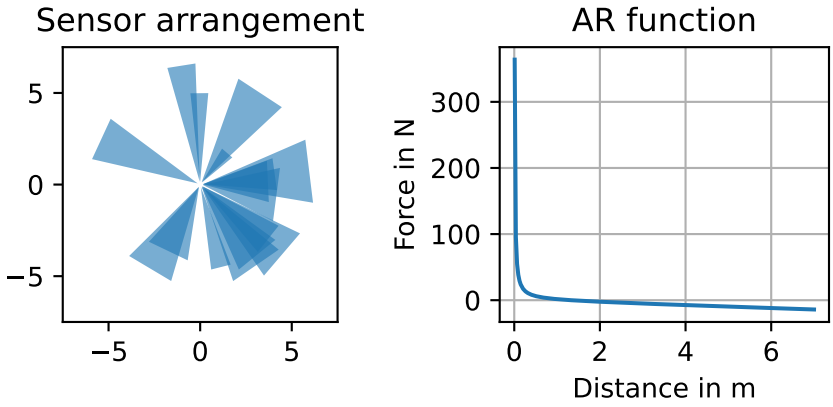


Figure 6.13.: Sensor configuration and plot of the attraction-repulsion function of the individual from ARI 5 10. The blue triangles represent the area in which the sensors can detect other copters.

Finally, an individual from ARIII 6 10 is inspected, only one of those individuals was included in the Pareto optimal set. ARIII 6 10 is another parameter set which showed good motion quality. With this individual, the copters never manage to follow the leader and the overall movement of the copters seems very chaotic, with many direction changes and no predominant direction the copters agreed on (see Figure 6.14). Despite that, this individual achieved a good motion quality, as the direction changes only occur in the first part of the simulation and the motion quality is measured towards the end.

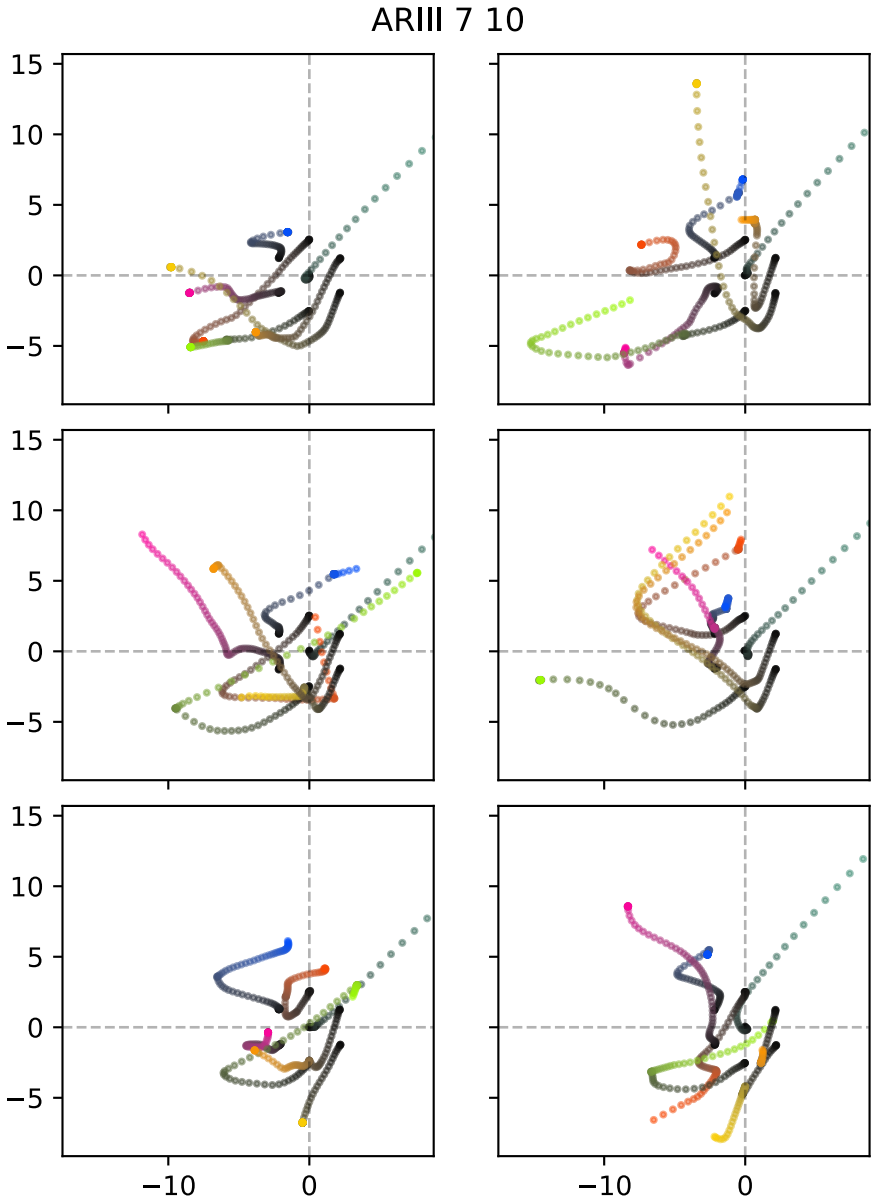


Figure 6.14.: Example flight paths of an individual from parameter set ARIII 6 10

Generally, the copters stay closer to the origin than the previous two individuals. This a probably a result of the comparatively very low repulsion force (see Figure 6.15) found in this individual. Furthermore, this individual has a fairly

uneven distribution of sensors, with most of them being oriented towards the top.

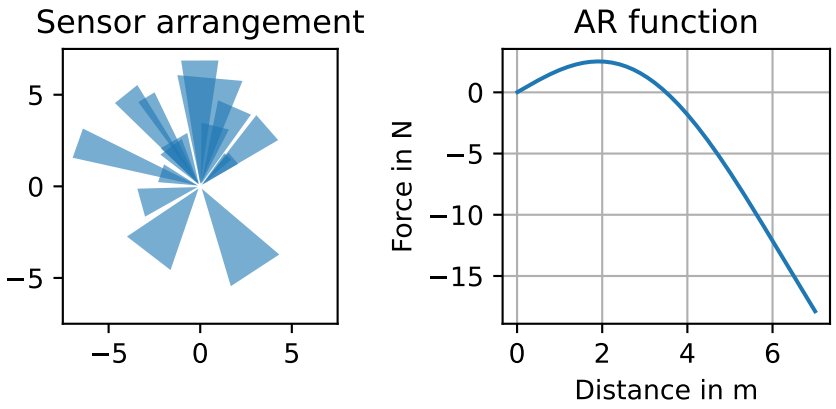


Figure 6.15.: Sensor configuration and plot of the attraction-repulsion function of the individual from ARIII 6 10. The blue triangles represent the area in which the sensors can detect other copters.

7. Conclusion and Future Work

This chapter at first answers the research questions listed in the introduction, and the addresses possible improvements and future work to be done on SPOC.

7.1. Conclusion

RQ1 Is it possible to co-optimize structural and behavioral parameters of a swarm of quadcopters performing an aggregation behavior using an evolutionary algorithm?

SPOC is definitely capable of producing good solutions regarding the specified objectives. Its solutions measurably outperform the reference individuals. But neither the reference solutions nor the produced solutions consistently show a behavior where the whole swarm moves as a unit in an aggregated fashion, often the swarm members break apart from the leader. Perhaps there is some part of the simulation framework which inhibits the display of such a behavior. It could be that the inherent latency of the system just does not work well with any of the three chose attraction repulsion parameters, or that the envisioned behavior is just notoriously difficult to achieve with the given setup.

RQ2 Is it possible to perform this optimization in a fully automatic fashion?

SPOC is almost fully automated, with only some intervention needed at the start and at the end. Initially, some parameters need to be set, like swarm size, number of sensors etc. And finally, the solutions need to be inspected in order to select an individual with an appropriate trade-off between the objectives, just like in other multi-objective optimizations. There are still some technical problems where some simulations get stuck and run into a timeout repeatedly, causing some individuals to not get properly evaluated. This does not necessarily require intervention because SPOC continues to function fairly well after such an occurrence because of the global archive that is used. For

further use of SPOC, however, this problem should be more closely investigated and fixed, as this would probably improve the results delivers by SPOC.

RQ3 Is it possible to detect any impact of the structural parameters on the behavioral parameters or vice versa?

These two parts are very intertwined and difficult to separate from each other; hence they are often co-evolved. The results show no immediate connection between some set of sensor parameters with another configuration of the attraction-repulsion function. Perhaps with some further statistical analysis a correlation could be found, but it is not clear whether this is possible.

RQ4 Can this method produce solutions which outperform solutions which are manually designed by an expert?

Neither the reference solutions nor the solutions produced by SPOC successfully perform the intended behavior, which makes it difficult to answer this question. When this is purely judged by the fitness of the solutions, then the answer is yes. But when inspecting the behavior more closely by looking at the flight paths, this is less clear. Some solutions sometimes come close to the intended behavior, but do not do this consistently. It is likely that the chosen fitness functions just do not properly encode the envisioned behavior. Perhaps there is a different set of fitness function with which SPOC would produce solutions which behave more like intended. This would likely include an adaption of the aggregation quality, which also measure distance to the swarm center.

7.2. Future Work

The swarm aggregation behavior used in this work is purely two-dimensional. A three-dimensional behavior would add a large amount of complexity and would likely require a larger number of sensors for any sort of aggregation to occur. This amount of complexity was just not feasible to add to SPOC at this time, but with some further work, it would certainly be possible in order to examine if SPOC also works on 3 dimensional behaviors. Aggregation quality and motion quality are easily adaptable to three dimensions, but it might also be worth to look at other fitness functions. Only few individuals managed to get close to the behavior that the fitness functions were supposed to represent, there might be other fitness functions which do a better job.

This work does not address the reality gap between the simulated copters and the FINken copters by which they are inspired. The masses of the components were measured from the FINken and the sonar sensor parameters are in ranges which are also achievable with the real hardware. But the sonar sensor implementation in the simulation is only an approximation of a real sonar sensors and the aerodynamics — while modelled with proper lift and drag forces — might still be very different from a real copter. As the swarm behavior of SPOC is implemented using Ardupilot it should be possible to use it as is on the real copters and compare the behavior to the simulation and tune the aerodynamics as well as the sonar sensors. This should be done in the future to enable SPOC to produce solutions which can be transferred to the real copters with more certainty.

As seen in Figure 6.6, a single individual produces a range of fitness values when evaluated multiple times. The method to deal with this in this work, was to introduce an additional age objective together with a rolling average filter on the fitness values, in order to gain certainty on the real fitness value of an individual with a higher certainty. This work does not evaluate the effectiveness of this method or compares it to other methods. But there certainly are other ways of dealing with this problem which should be evaluated. One way could be to change the dominance operator for the selection operator to one that uses the Mann Whitney U test to compare the distributions of fitness values of two individuals. Another could be to use a penalty function in addition to the rolling average, in order to prefer individuals with more evaluations, which would eliminate the age objective.

This work used a maximum swarm size of 7 due to technical limitations, which is a rather small swarm size. With higher computational power or more optimization, it might be possible to try larger swarm sizes in order to evaluate the scalability of this approach.

A. Experiments and Results

Table A.1.: Table of all parameter combinations used for hyperparameter exploration

Parameter Set	Pop. Size	RER	MR	Mσ
1	10	3	0.2	0.1
2	10	3	0.2	0.25
3	10	3	0.2	0.5
4	10	3	0.3	0.1
5	10	3	0.3	0.25
6	10	3	0.3	0.5
7	10	3	0.4	0.1
8	10	3	0.4	0.25
9	10	3	0.4	0.5
10	10	5	0.2	0.1
11	10	5	0.2	0.25
12	10	5	0.2	0.5
13	10	5	0.3	0.1
14	10	5	0.3	0.25
15	10	5	0.3	0.5
16	10	5	0.4	0.1
17	10	5	0.4	0.25
18	10	5	0.4	0.5
19	16	5	0.2	0.1
20	16	5	0.2	0.25
21	16	5	0.2	0.5
22	16	5	0.3	0.1
23	16	5	0.3	0.25
24	16	5	0.3	0.5
25	16	5	0.4	0.1

Table A.1.: continued

Parameter Set	Pop. Size	RER	MR	Mσ
26	16	5	0.4	0.25
27	16	5	0.4	0.5
28	16	8	0.2	0.1
29	16	8	0.2	0.25
30	16	8	0.2	0.5
31	16	8	0.3	0.1
32	16	8	0.3	0.25
33	16	8	0.3	0.5
34	16	8	0.4	0.1
35	16	8	0.4	0.25
36	16	8	0.4	0.5
37	22	7	0.2	0.1
38	22	7	0.2	0.25
39	22	7	0.2	0.5
40	22	7	0.3	0.1
41	22	7	0.3	0.25
42	22	7	0.3	0.5
43	22	7	0.4	0.1
44	22	7	0.4	0.25
45	22	7	0.4	0.5
46	22	11	0.2	0.1
47	22	11	0.2	0.25
48	22	11	0.2	0.5
49	22	11	0.3	0.1
50	22	11	0.3	0.25
51	22	11	0.3	0.5
52	22	11	0.4	0.1
53	22	11	0.4	0.25
54	22	11	0.4	0.5

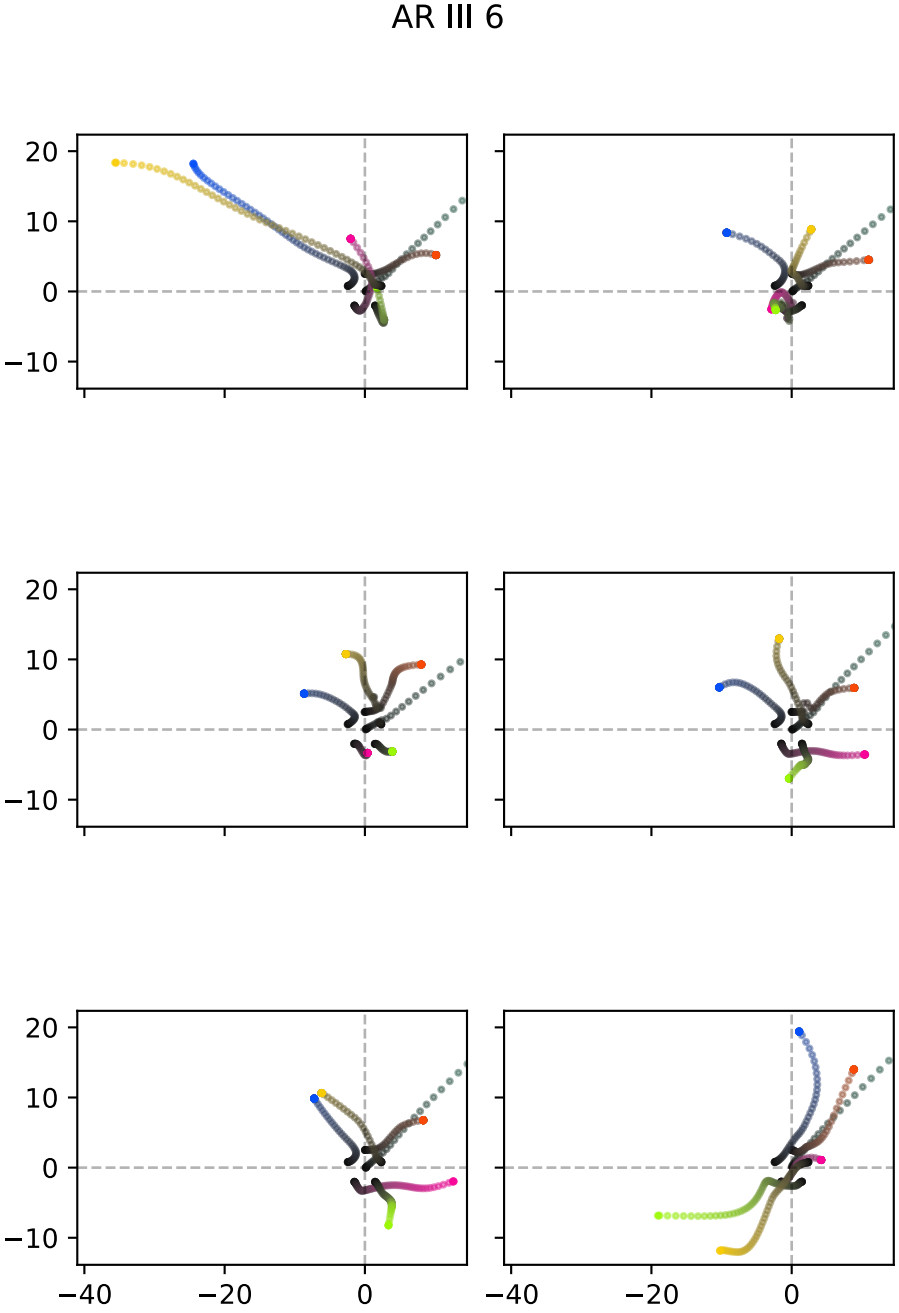


Figure A.1.: Example flight paths of the reference individuals with swarm size 6 and attraction repulsion function III. Top-down view with scale in meters.

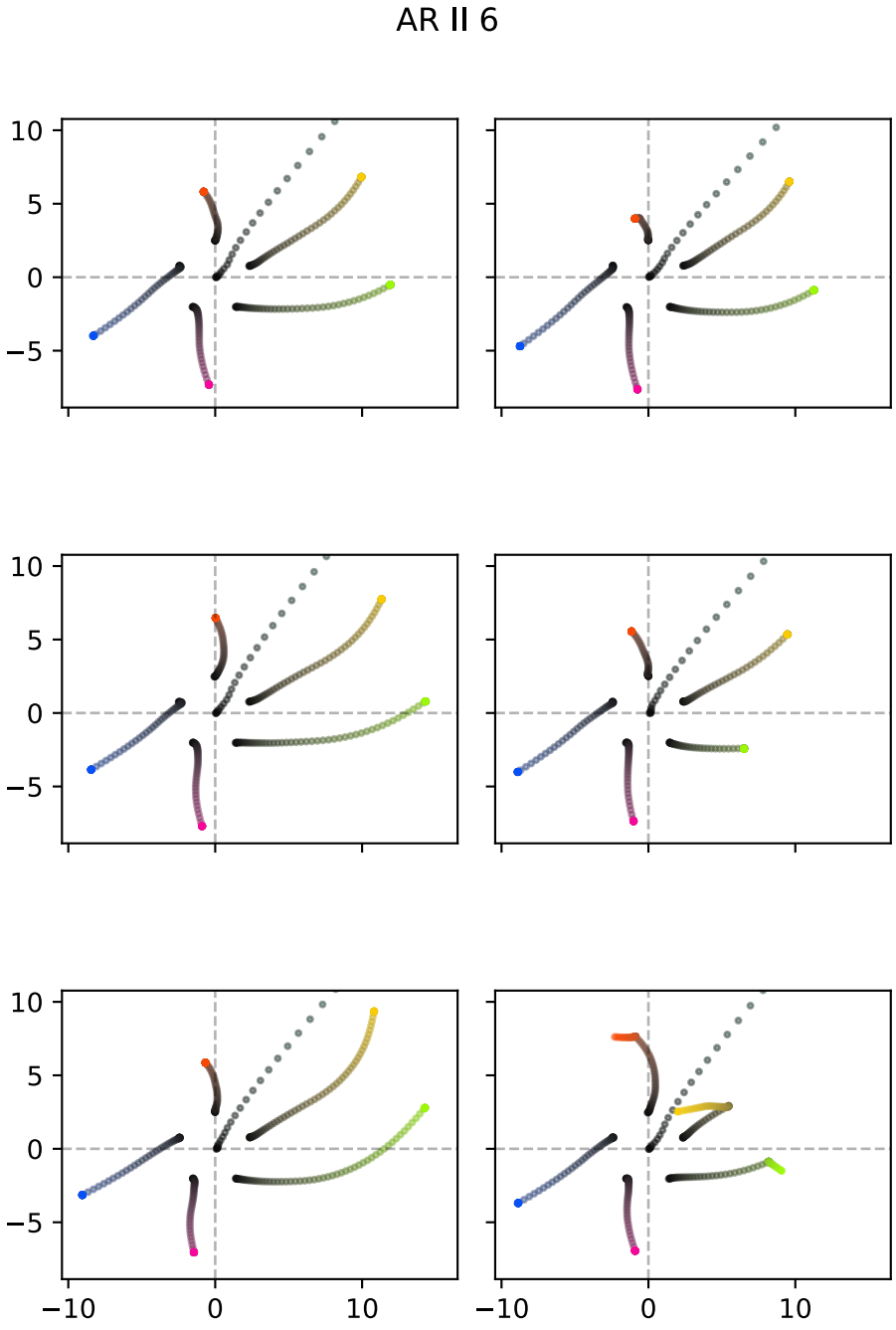


Figure A.2.: Example flight paths of the reference individuals with swarm size 6 and attraction repulsion function II. Top-down view with scale in meters.

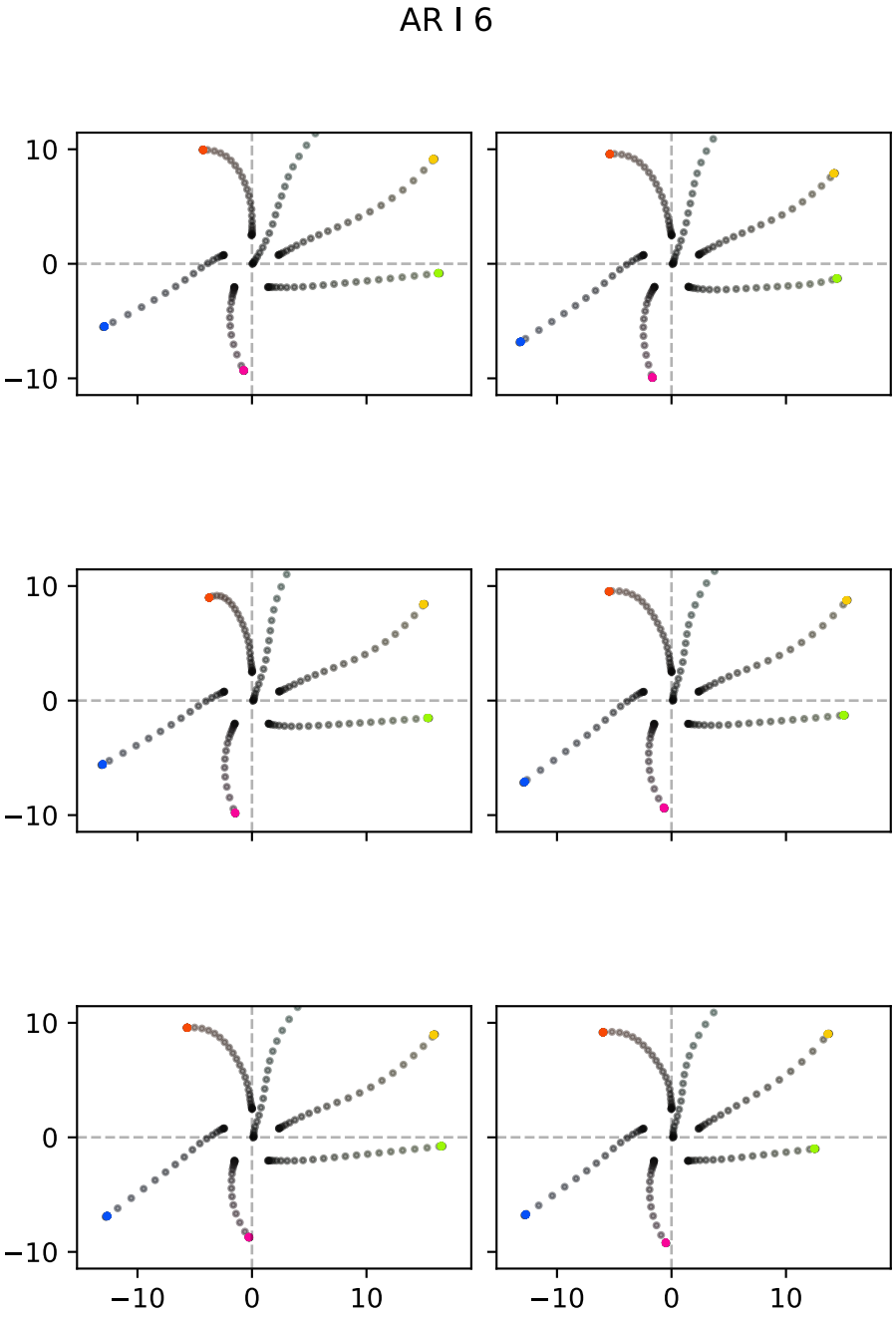


Figure A.3.: Example flight paths of the reference individuals with swarm size 6 and attraction repulsion function I. Top-down view with scale in meters.

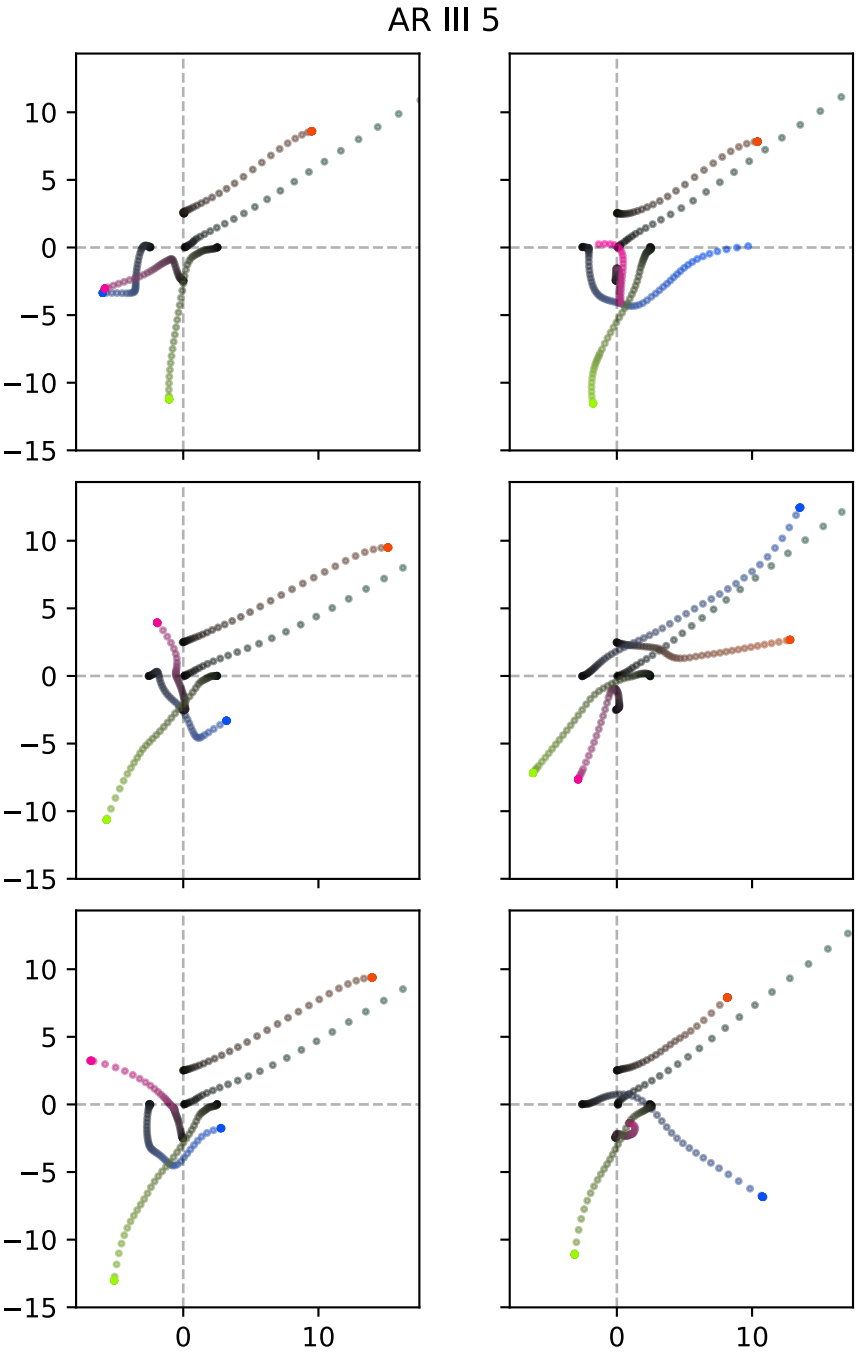


Figure A.4.: Example flight paths of the reference individuals with swarm size 5 and attraction repulsion function III. Top-down view with scale in meters.

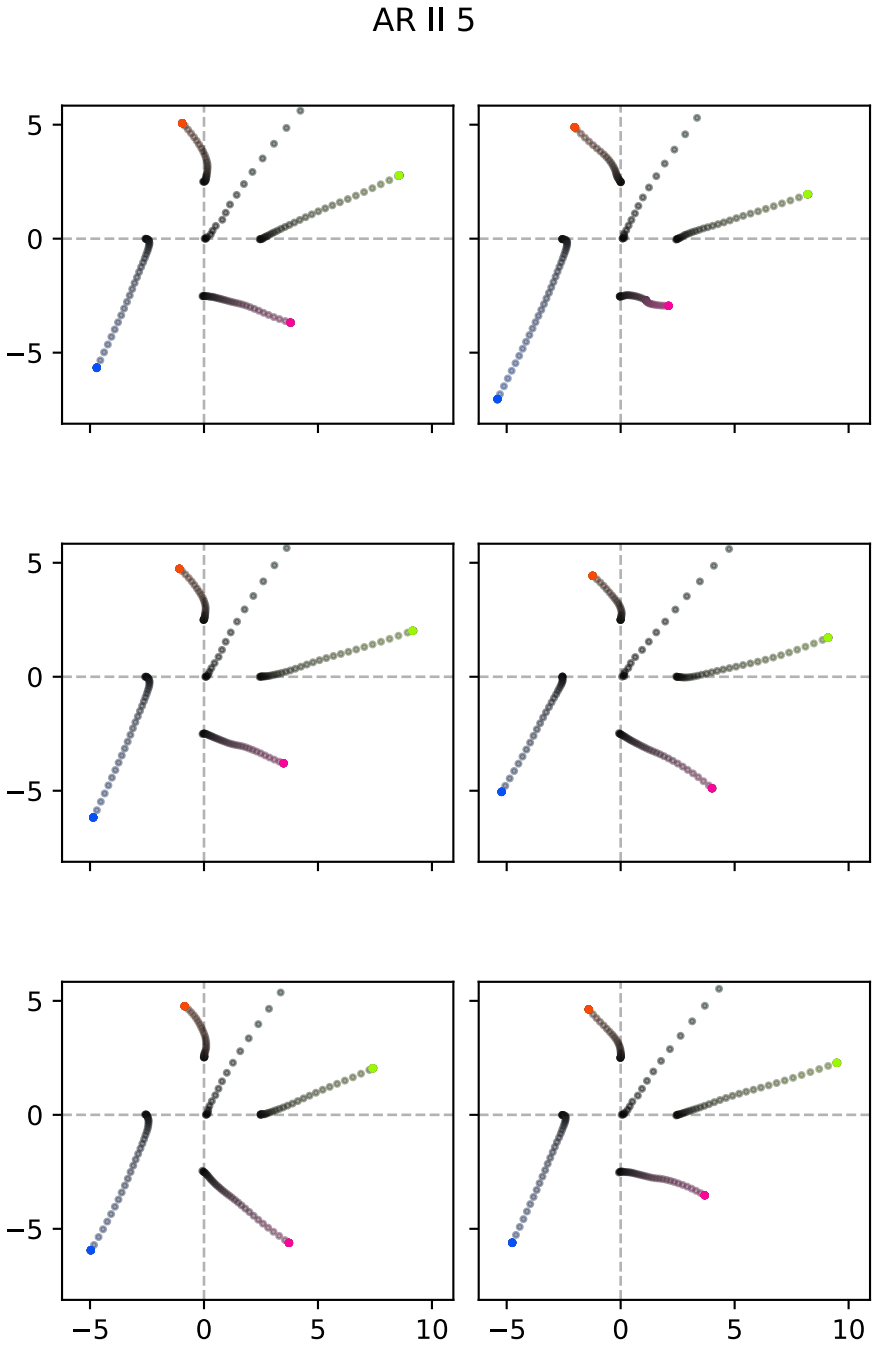


Figure A.5.: Example flight paths of the reference individuals with swarm size 5 and attraction repulsion function II. Top-down view with scale in meters.

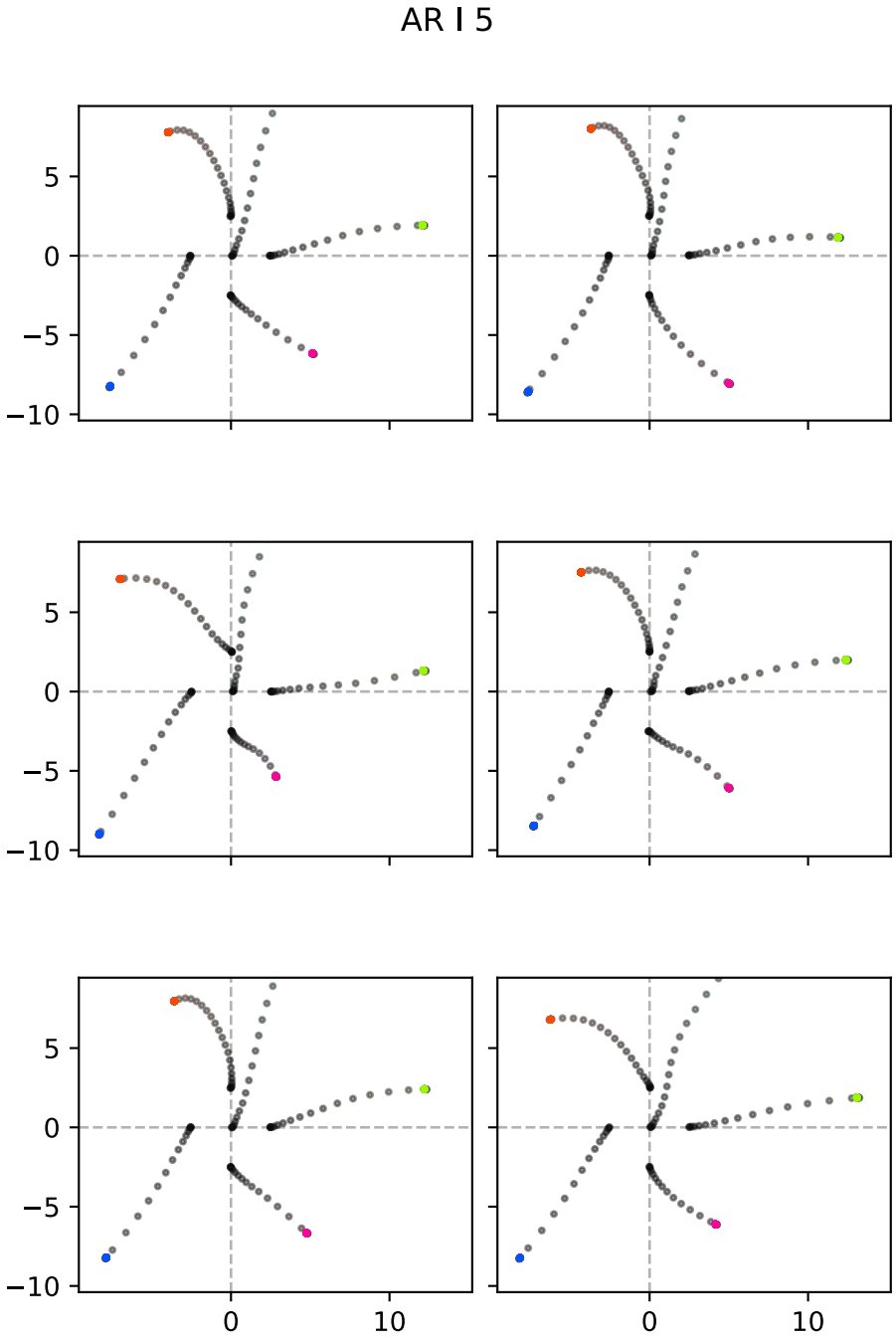


Figure A.6.: Example flight paths of the reference individuals with swarm size 5 and attraction repulsion function I. Top-down view with scale in meters.

Figure A.7.: Example flight paths of the non-dominated solutions produced by the EA. Each row contains 5 runs of the same individual. The labeling on the left denotes the used AR function, swarm size and initial parameter range for the AR function. Top-down view with scale in meters.

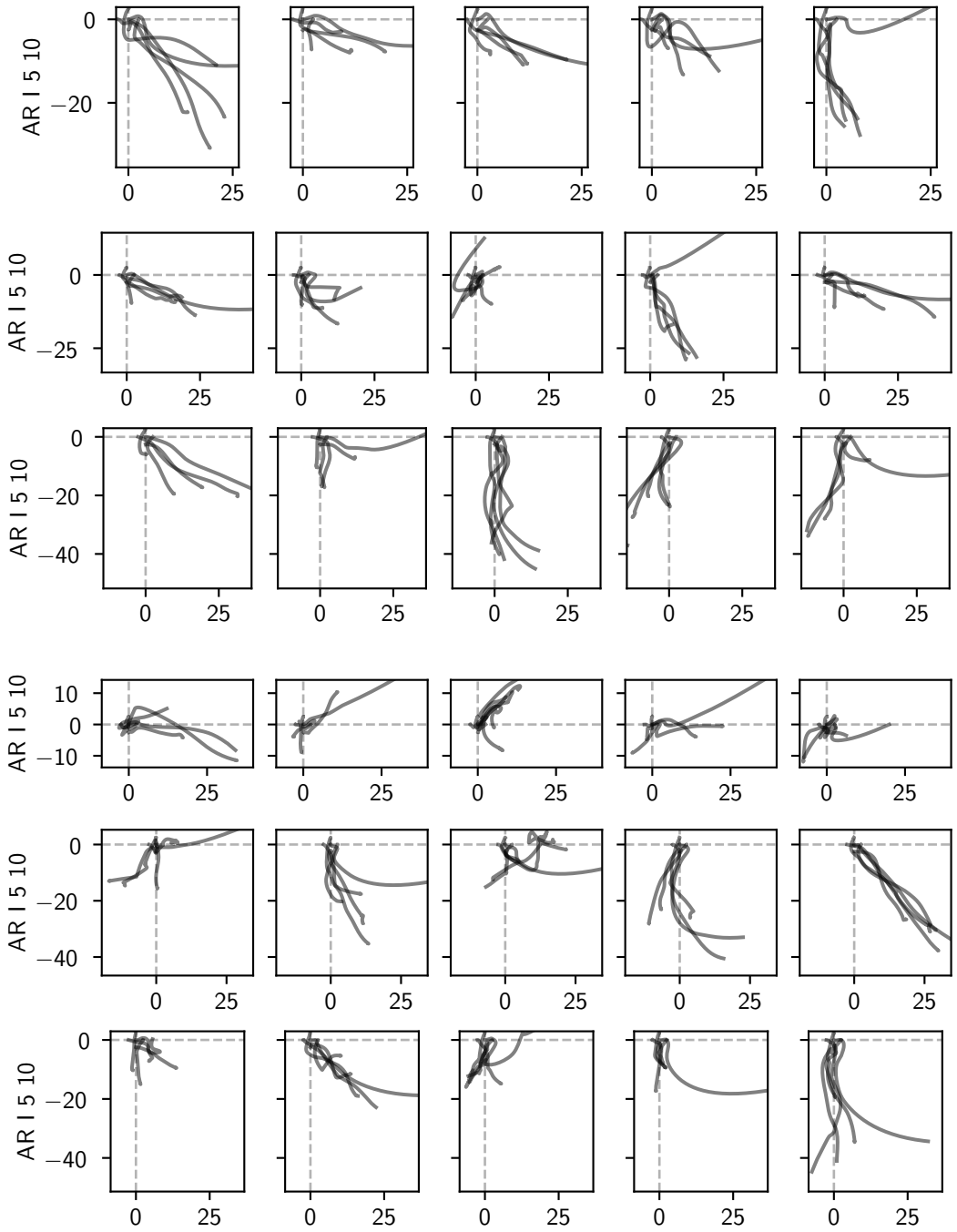


Figure A.7.: (cont.)

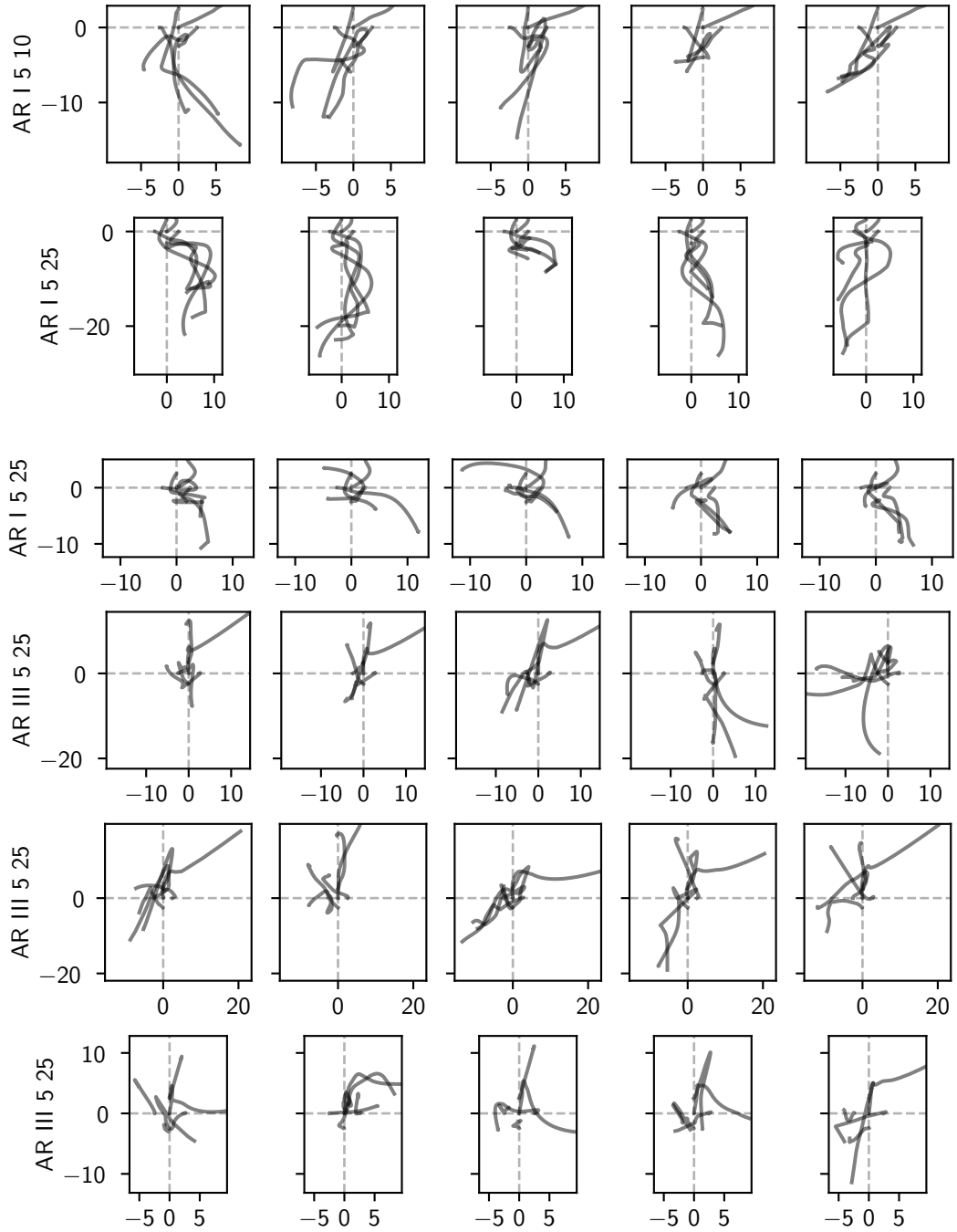


Figure A.7.: (cont.)

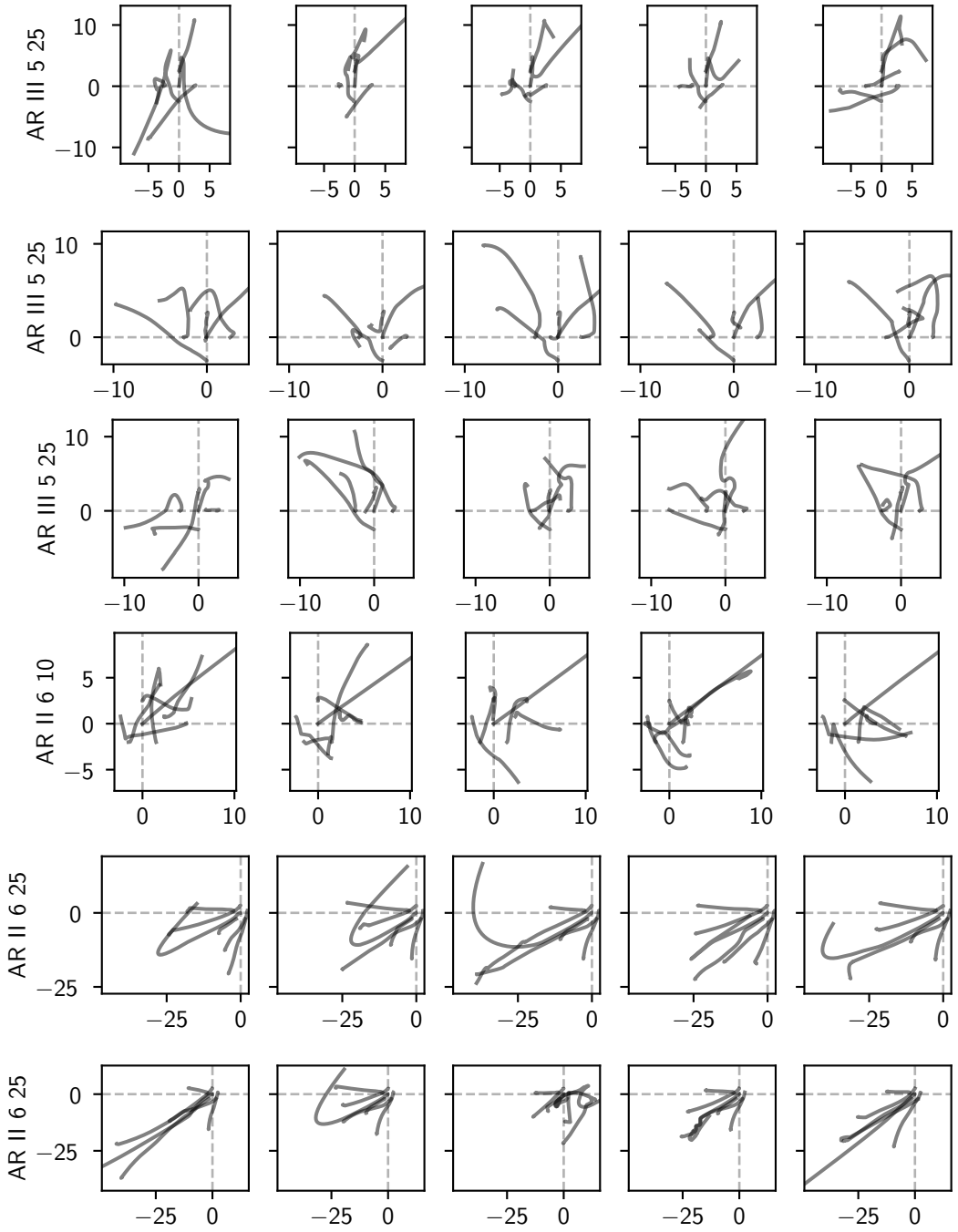
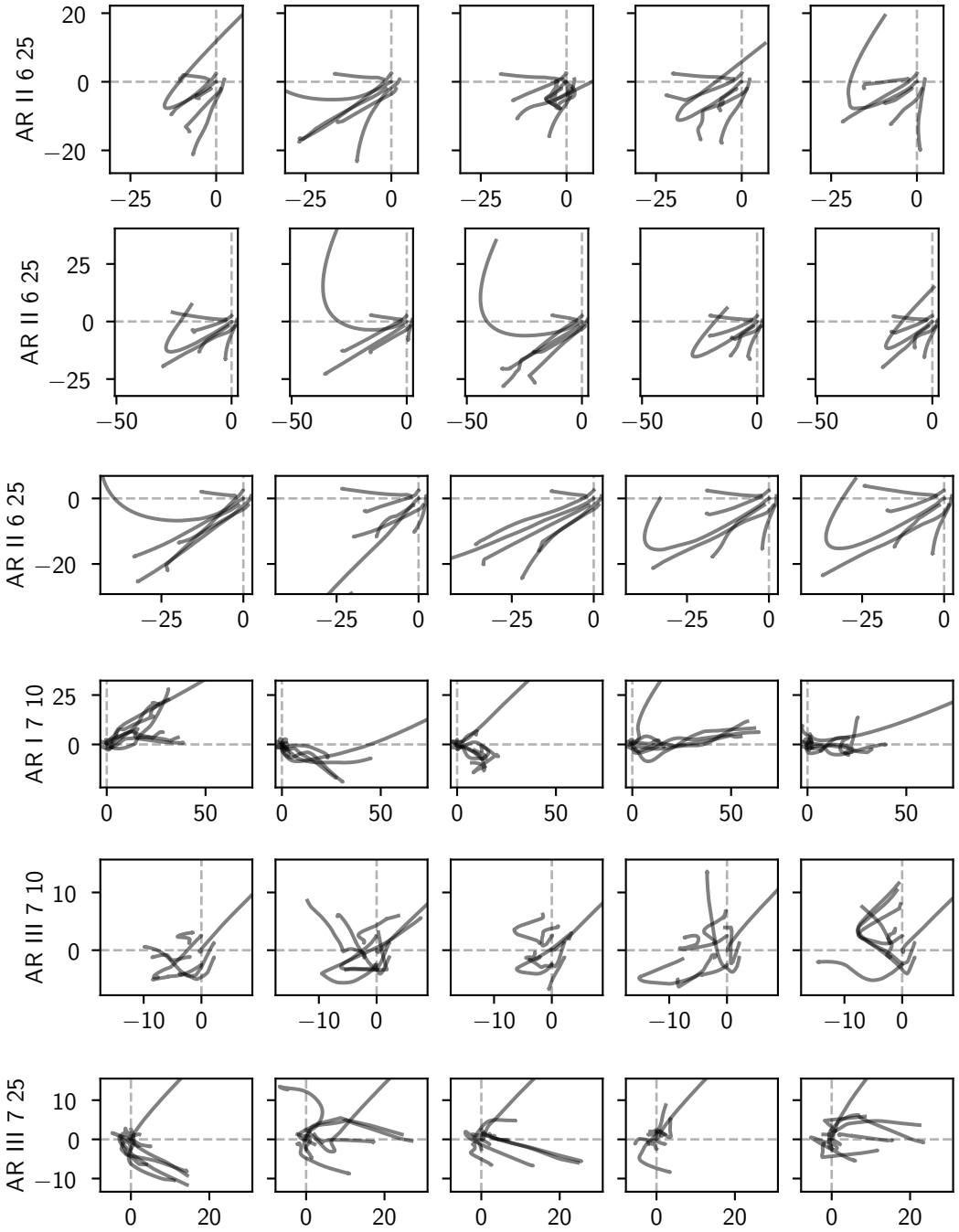


Figure A.7.: (cont.)



Bibliography

[1] R. J. Alattas, S. Patel, and T. M. Sobh. "Evolutionary modular robotics: survey and analysis". en. In: *Journal of intelligent & robotic systems* 95.3-4 (Sept. 2019), pp. 815–828. ISSN: 0921-0296, 1573-0409. DOI: 10.1007/s10846-018-0902-9. (Visited on 09/05/2022).

[2] G. Barlow, C. Oh, and E. Grant. "Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming". In: *IEEE Conference on Cybernetics and Intelligent Systems, 2004.* Vol. 2. Dec. 2004, pp. 689–694. DOI: 10.1109/ICCIS.2004.1460671.

[3] G. Beni. "Swarm intelligence". en. In: *Complex social and behavioral systems*. Ed. by M. Sotomayor, D. Pérez-Castrillo, and F. Castiglione. New York, NY: Springer US, 2020, pp. 791–818. ISBN: 978-1-07-160367-3 978-1-07-160368-0. DOI: 10.1007/978-1-0716-0368-0_530. (Visited on 09/06/2022).

[4] G. Beni and J. Wang. "Swarm intelligence in cellular robotic systems". en. In: *Robots and biological systems towards a new bionics?* Ed. by P. Dario, G. Sandini, and P. Aebischer. NATO ASI Series. Berlin, Heidelberg: Springer, 1993, pp. 703–712. ISBN: 978-3-642-58069-7. DOI: 10.1007/978-3-642-58069-7_38.

[5] C. Blum and D. Merkle, eds. *Swarm intelligence: introduction and applications*. en. Natural computing series. Berlin Heidelberg: Springer, 2008. ISBN: 978-3-540-74088-9.

[6] M. D. Bugajska and A. C. Schultz. *Co-Evolution of Form and Function in the Design of Autonomous Agents: Micro Air Vehicle Project*. en. Tech. rep. ADA480643. Section: Technical Reports. Defense Technical Information Center.

[7] A. J. Clark, X. Tan, and P. K. McKinley. "Evolutionary multiobjective design of a flexible caudal fin for robotic fish". en. In: *Bioinspiration & biomimetics* 10.6 (Nov. 2015), p. 065006. ISSN: 1748-3190. DOI: 10.1088/1748-3190/10/6/065006. (Visited on 09/15/2022).

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: nsga-ii". In: *Ieee transactions on evolutionary computation* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017.

[9] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, and L. M. Gambardella. "Evolving self-organizing behaviors for a swarm-bot". en. In: *Autonomous robots* 17.2 (Sept. 2004), pp. 223–245. ISSN: 1573-7527. DOI: 10.1023/B:AURO.0000033973.24945.f3. (Visited on 09/06/2022).

[10] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. "DEAP: evolutionary algorithms made easy". In: *Journal of machine learning research* 13 (July 2012), pp. 2171–2175.

[11] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari. "AutoMoDe: a novel approach to the automatic design of control software for robot swarms". en. In: *Swarm intelligence* 8.2 (June 2014), pp. 89–112. ISSN: 1935-3820. DOI: 10.1007/s11721-014-0092-4. (Visited on 09/13/2022).

[12] V. Gazi and K. Passino. "Stability analysis of swarms". In: *Ieee transactions on automatic control* 48.4 (Apr. 2003). Conference Name: IEEE Transactions on Automatic Control, pp. 692–697. ISSN: 1558-2523. DOI: 10.1109/TAC.2003.809765.

[13] B. von Haller, A. J. Ijspeert, and D. Floreano. "Co-evolution of structures and controllers for neubot underwater modular robots". In: *Ecal* (2005). DOI: 10.1007/11553090_20.

[14] H. Hamann. *Swarm Robotics: A Formal Approach*. en. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-74526-8 978-3-319-74528-2. DOI: 10.1007/978-3-319-74528-2. URL: http://link.springer.com/10.1007/978-3-319-74528-2 (visited on 08/29/2022).

[15] G. S. Hornby and J. B. Pollack. "Body-brain co-evolution using L-systems as a generative encoding". In: *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. GECCO'01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., July 2001, pp. 868–875. ISBN: 978-1-55860-774-3. (Visited on 09/18/2022).

[16] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin. "Fault-tolerant cooperative navigation of networked UAV swarms for forest fire monitoring". en. In: *Aerospace science and technology* 123 (Apr. 2022), p. 107494. ISSN: 1270-9638. DOI: 10.1016/j.ast.2022.107494. (Visited on 10/18/2022).

[17] R. Kruse, S. Mostaghim, C. Borgelt, C. Braune, and M. Steinbrecher. *Computational Intelligence: A Methodological Introduction*. en. Texts in Computer Science. Cham: Springer International Publishing, 2022. ISBN: 978-3-030-42226-4 978-3-030-42227-1. DOI: 10.1007/978-3-030-42227-1. (Visited on 08/05/2022).

[18] D. Mader, R. Blaskow, P. Westfeld, and C. Weller. "Potential of uav-based laser scanner and multispectral camera data in building inspection". In: *The international archives of the photogrammetry, remote sensing and spatial information sciences* XLI-B1 (2016), pp. 1135–1142. DOI: 10.5194/isprs-archives-XLI-B1-1135-2016. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B1/1135/2016/.

[19] M. Mohid and J. F. Miller. "Evolving robot controllers using carbon nanotubes". en. In: *07/20/2015-07/24/2015*. The MIT Press, July 2015, pp. 106–113. ISBN: 978-0-262-33027-5. DOI: 10.7551/978-0-262-33027-5-ch025. (Visited on 09/15/2022).

[20] J. M. Moore and P. K. McKinley. "Evolution of an amphibious robot with passive joints". In: *2013 IEEE Congress on Evolutionary Computation*. ISSN: 1941-0026. June 2013, pp. 1443–1450. DOI: 10.1109/CEC.2013.6557733.

[21] J. Mwaura and E. Keedwell. "Evolving robot sub-behaviour modules using gene expression programming". en. In: *Genetic programming and evolvable machines* 16.2 (June 2015), pp. 95–131. ISSN: 1389-2576, 1573-7632. DOI: 10.1007/s10710-014-9229-x. (Visited on 09/16/2022).

[22] G. B. Parker and P. J. Nathan. "Co-evolution of sensor morphology and control on a simulated legged robot". In: *2007 International Symposium on Computational Intelligence in Robotics and Automation*. June 2007, pp. 516–521. DOI: 10.1109/CIRA.2007.382874.

[23] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, and I. Moscholios. "A compilation of UAV applications for precision agriculture". en. In: *Computer networks* 172 (May 2020), p. 107148. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2020.107148. (Visited on 10/18/2022).

[24] E. Şahin and W. M. Spears, eds. *Swarm robotics: SAB 2004 international workshop, Santa Monica, CA, USA, July 17, 2004: revised selected papers*. en. Lecture notes in computer science, State-of-the-art survey 3342. Meeting Name: International Conference on Simulation of Adaptive Be-

havior OCLC: ocm57597119. Berlin ; New York: Springer, 2005. ISBN: 978-3-540-24296-3.

[25] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek. "Autonomous UAV surveillance in complex urban environments". In: *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*. Vol. 2. Sept. 2009, pp. 82–85. DOI: 10.1109/WI-IAT.2009.132.

[26] V. Trianni. *Evolutionary Swarm Robotics*. en. Ed. by J. Kacprzyk. Vol. 108. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-77611-6 978-3-540-77612-3. DOI: 10.1007/978-3-540-77612-3. (Visited on 09/06/2022).

[27] V. Trianni and M. Lopez-Ibanez. *Advantages of multi-objective optimisation in evolutionary robotics: survey and case studies*. Technical TR/IRIDIA/2014-014. IRIDIA, Institut de Recherches Interdisciplinaires et de Developpements en Intelligence Artificielle Universite Libre de Bruxelles, Nov. 2014, p. 36. (Visited on 09/19/2022).

[28] T. Yasuda, K. Ohkura, T. Nomura, and Y. Matsumura. "Evolutionary swarm robotics approach to a pursuit problem". In: *2014 IEEE Symposium on Robotic Intelligence in Informationally Structured Space (RiiSS)*. Dec. 2014, pp. 1–6. DOI: 10.1109/RIISS.2014.7009182.

Declaration of Authorship

I hereby declare that this thesis was created by me and me alone using only the stated sources and tools.

Nico Winkelsträter Magdeburg, October 25, 2022