Fabian Witt

Energy Aware Particle Swarm Optimization as Search Mechanism for Aerial Micro-Robots

Faculty of Computer Science
Intelligent Systems Department

Master Thesis

Energy Aware Particle Swarm Optimization as Search Mechanism for Aerial Micro-Robots

Autor: Fabian Witt

Professor: Prof. Dr. Sanaz Mostaghim

Examiner: Prof. Dr. Sanaz Mostaghim

Examiner: Prof. Dr. Heiko Hamann

Summer term 2016

Fabian Witt: *Energy Aware Particle Swarm Optimization as Search Mechanism for Aerial Micro-Robots*
Otto-von-Guericke University
Magdeburg, 2016.

This thesis presents the Energy Aware PSO (EAPSO) as a search mechanism for aerial micro-robots with limited energy capacity. The proposed model is an extension of the search concept of Particle Swarm Optimization (PSO) that additionally considers the energy levels of the individuals for an efficient movement. One major contribution of this thesis is that the energy efficiency results from a multi-criteria decision making process performed by the individuals. The energy consumption model in EAPSO is adapted from a real hardware scenario and has been tested on three known landscapes which are very similar to search terrains by the aerial micro-robots. The results show that EAPSO can reduce the total energy consumption of the swarm with negligible degradation of the search results.

The proposed approach works as follows: each individual has the ability to choose between different actions (start, land and fly) and neighborhood sizes. The neighborhood includes the other individuals in the swarm. We use the neighborhood to select the global best individual for the PSO formula. With this process, each individual has the ability to decide between different flights. The decision is made based on two objectives, profit in terms of the overall gain in search process and cost in terms of the energy consumption. The values are weighted by the risk value of the individual.

Contents

List of Figures

List of Tables

List of Algorithms

Statutory Declaration

I assure that this thesis is a result of my personal work and that no other than the indicated aids have been used for its completion. Furthermore, I assure that all quotations and statements that have been inferred literally or in a general manner from published or unpublished writings are marked as such.

Beyond this I assure that the work has not been used, neither completely nor in parts, to pass any previous examination.

first and last name	city, date and signature

1 Introduction

Swarm Robotics (SR) has been the subject of research for almost over a decade. The major properties concern a large number of simple robotic systems and simple rules. The swarm is supposed to collectively learn a predefined given task [Trianni, 2008]. One important challenge in micro-robotic systems is managing the energy resources, which is a crucial aspect in accomplishing a task by such autonomous systems [O'Hara et al., 2006]. This is notably evident with small aerial robots, which have severely limited battery of typically 10 to 15 minutes [Roberts et al., 2008, Valenti et al., 2007]. Different to ground robots, aerial robots have significantly different energy dynamics, require substantially more energy to locomote [Roberts et al., 2008, Stirling and Floreano, 2013], and the small payload entails reduced sensing and processing capabilities.

The proposed method introduces a new PSO-based search mechanism called Energy Aware PSO (EAPSO) which additionally considers the amount of energy consumption for each individual. In contrast to the standard PSO, the individuals estimate the amount of required energy for moving to the next position and decide about the movement by considering the trade-off between profit and energy consumption. The thesis presents various models for multi-criteria decision making and the experiments and comparisons illustrate that with EAPSO the individuals are able to save a large amount of energy without degrading the quality of search. EAPSO is modeled based on energy consumption of a real hardware scenario. However, the presented work in this thesis only considers the search mechanism and has met several assumptions. Decision making in the swarms has been studied in other contexts presented by [Valentini et al., 2015, Wahby et al., 2015] in which the authors investigate different robot controllers for collective decision-making in the context of the speed-versus-accuracy trade-off. The proposed approach in this thesis is different from collective decision making in swarms as we let the swarm members decide on their own by considering the local information.

1.1 State of the Art

Dealing with limited energy levels has been addressed in the literature, for instance, an algorithm for indoor aerial swarm search that exploits the ability of flying robots to attach to ceilings and saves energy was developed by [Roberts et al., 2008] and [Stirling et al., 2010]. A novel strategy was studied by [Stirling and Floreano, 2013] which controls the density of flying robots. They illustrate an efficient way of reducing swarm energy costs while maintaining a rapid search. Other approaches consider Underwater robots [Amory et al., 2014] saving energy by staying on the surface or forming a V-formation [Amory et al., 2013]. These works are among the first publications addressing the trade-off. Time and energy were previously either examined independently, or only with multi-objective functions that mask trends in the individual metrics. Prior work also usually neglected the energy consumption of sensors and processors [Mei et al., 2005]. In many scenarios a rapid deployment is desired. However, time and energy are not independent, and often there is a trade-off as indicated by [Moscibroda et al., 2006, Hayes, 2002, Mei et al., 2006]. An additional aspect in swarm robotics is controlling deployment into unknown environments. If robots deploy to unnecessary locations, energy is wasted. Conversely, if an area receives insufficient robots the task may be unachievable or the performance is reduced. Nevertheless, a crucial feature is the autonomy of robotic systems, i.e., the individual robots need to be able to make decisions on their own by considering own and the performance of the other swarm members. Another challenge is the charging of the aerial robots. For this, [Mulgaonkar and Kumar, 2014] introduces a system of autonomous charging stations for a team of Micro Aerial Vehicles (MAVs). The presented MAVs are able to recharge their on-board batteries without any human intervention. An extension of this approach is to work with unmanned ground vehicles (UGVs) as mobile charging stations [Mulgaonkar and Kumar, 2014, Michael et al., 2012].

1.2 Research Goals and Specific Objectives

The main goal of this thesis is to propose a new model for a search mechanism in a swarm of small flying robots (aerial micro-robots) which is based on Particle Swarm Optimization method. The major achievements of this thesis can

be summarized as follows:
To use the standard PSO method as a search mechanism for aerial micro-robots, a model of the robots is required. This includes the modeling of the movements, as well as the mapping of the movement in discrete time steps. Therefore, the first objective is to build a movement model, with different actions like fly, start, and land, for the individuals.

Objective 1: Modeling a general movement behavior for the individuals in a swarm

In order to analyze the energy consumption of different methods, we need a model that describes the needed energy of the copter for different actions. Therefore, the second objective is to provide a model of energy consumption for the copters, based on our micro aerial robot FINken-III.

Objective 2: Building a simplified model of energy consumption for the individuals, based on FINken-III

For an energy efficient optimization, it is necessary to integrate the energy into the standard PSO method. To achieve this, the third objective deals with the integration of the energy efficient decision making process into the conventional PSO method. The goal is, to present an algorithm, such that the individuals are able to save energy without degrading the quality of search.

Objective 3: PSO-based search mechanism which considers the amount of energy consumption for each individual

The fourth objective of this thesis concerns the implementation of the EAPSO approach. The implementation includes the movement model, the simplified model of energy consumption and the EAPSO model.

Objective 4: Implementation of the Energy Aware PSO approach for evaluation and future projects

The last objective cares about the evaluation of the various models for multi-criteria decision making. The aim is to show, whether the methods work and which properties they have.

> **Objective 5**: Evaluation of different EAPSO methods, to show whether the methods work out as desired

1.3 Document Organization

This thesis is organized as follows. Chapter 2 reviews the basic concepts of Particle Swarm Optimization (PSO) in Section 2.1, multi-criteria decision making in Section 2.2 and provides a description of our micro aerial robot FINken-III in Section 2.3. Afterwards, we introduce our model in Chapter 3. The structure is as follows. We show our movement model in Section 3.1, outline the new discrete time flight concept in Section 3.2, and describe the neighborhood topology in Section 3.3. Afterwards, we show the concept of the energy computations for our FINken-III robots in Section 3.4. Next, we demonstrate our new Energy Aware PSO method in Section 3.5, the selection and multi-criteria decision making in Section 3.6, and the profit approximation in Section 3.7. The implementation is shown in Chapter 4. Subsequently, Chapter 5 contains the evaluation of the method. Section 5.1 describes the experiments and Section 5.2 gives an explanation of the simulation environment. In Section 5.3, the results of the individual methods is shown. Finally, the thesis is summarized in Chapter 6 and an outlook is given.

2 Background

In this section, we describe the background about Particle Swarm Optimization, multi-criteria decision making and our micro aerial robot FINken-III. These three areas form the basis of our proposed model.

2.1 Particle Swarm Optimization (PSO)

Introduced by Kennedy and Eberhart 1995 [Kennedy and Eberhart, 1995], the first approach was described as population-based stochastic search and optimization process. In general, a swarm is defined as an apparently disorganized collection of moving individuals that tend to cluster together. The movement of a single individual, however, seems to be random. These behaviors can be observed by bird flocks or fish schools, which are searching for some targets (e.g., food). These kinds of swarms are self-organized and lead to the definition of an intelligence swarm. With the definition of an intelligence swarm as "a population of interacting individuals that optimizes a function or goal by collectively adapting to the local and/or global environment" [Kiranyaz et al., 2014], the functionality of PSO can be described.

> The behavior of a single organism in a swarm is often insignificant, but their collective and social behavior is of paramount importance. [Kiranyaz et al., 2014, p. 45]

This statement reflects the idea of Particle Swarm Optimization. The PSO approach considers a swarm, consisting of moving individuals or particles. Each particle has the goal to optimize the given cost-function. In the optimization process itself, each particle is moving through the search space and the position serves as input for the cost-function.

To reach the optimal solution, a swarm of N individuals in a n-dimensional space is considered, defined by S [Kennedy and Eberhart, 2001]. Each individual i has a position $\vec{x}_i(t) \in S$ and a velocity $\vec{v}_i(t)$ at time step t. The

individuals move in the search space by considering three factors: their own velocity vector at $t-1$, their own best obtained position $\vec{P}_{best}$ and the position of the globally (or locally) best individual from the population $\vec{x}_g$:

$$\vec{v}_i(t+1) = w\vec{v}_i(t) + C_1\phi_1(\vec{P}_{best} - \vec{x}_i(t)) + C_2\phi_2(\vec{x}_g - \vec{x}_i(t)) \quad (2.1)$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \quad (2.2)$$

Where ϕ_1 and ϕ_2 are two random vectors $\in [0,1]^n$. C_1 and C_2 are constants and determine the attraction rates. The globally best individual $\vec{x}_g$ can be defined using different communication topologies which is known to have a large impact on the convergence rate of PSO [Engelbrecht, 2005]. As selecting the globally best solution (leader) defines the amount of distance an individual might fly (depending on the random value ϕ_1), the topology can implicitly influence the energy consumption of the individuals. Ideally, the closer the globally best solution is located, the less energy is required to reach that point.

2.2 Multi-Criteria Decision Making

Multi-criteria decision making methods usually involve several conflicting objective functions $f_i(\vec{x})$ for $i = 1, \cdots, m$ and $\vec{x} \in S$ which have to be optimized at the same time. The solution of multi-objective optimization problems is usually a set of so called Pareto-optimal solutions from which the user has to select one. A solution $\vec{x}^*$ is called Pareto-optimal for minimization problems, if there is no other solution $\vec{x}'$ in the search space S so that:

$\forall i : f_i(\vec{x}') \leq f_i(\vec{x}^*)$ and $\exists j : f_j(\vec{x}') < f_j(\vec{x}^*)$.

Accordingly, we can use the same definition to compare the solutions. In this case, a solution $\vec{x}_1$ *dominates* another solution $\vec{x}_2$ (denoted by $\vec{x}_1 \prec \vec{x}_2$), if:

$\forall i : f_i(\vec{x}_1) \leq f_i(\vec{x}_2)$ and $\exists j : f_j(\vec{x}_1) < f_j(\vec{x}_2)$.

The solutions which do not mutually dominate each other are called non-dominated solutions. Selecting one of the Pareto-optimal (or non-dominated) solutions depends on the preferences of the user and can vary accordingly [Miettinen, 1999, Purshouse et al., 2014]. Weighted sum approach [Marler and Arora, 2010, Grodzevich and Romanko, 2006] is known to be the most straight forward mechanism to incorporate the preferences of the user in a weight vector $(w_1, \cdots, w_m)$ where $\sum_{i=1}^{m} w_i = 1$. Each w_i indicates

the relative preference towards the objective function i. Different vector values lead to different selection preferences.

2.3 Micro Aerial Robot FINken-III

In this Section we describe the underlying hardware platform FINken-III[1]. Figure 2.1 shows the copter FINken-III with the current sensors. The copters are developed to allow an autonomous behavior. A great attention is paid to the independence of external sensors. The copters send all sensor data to a ground station. This is only for monitoring. The copter can act without a connection to the ground station. In addition, the copter can be tracked via built-in LED and a camera system.

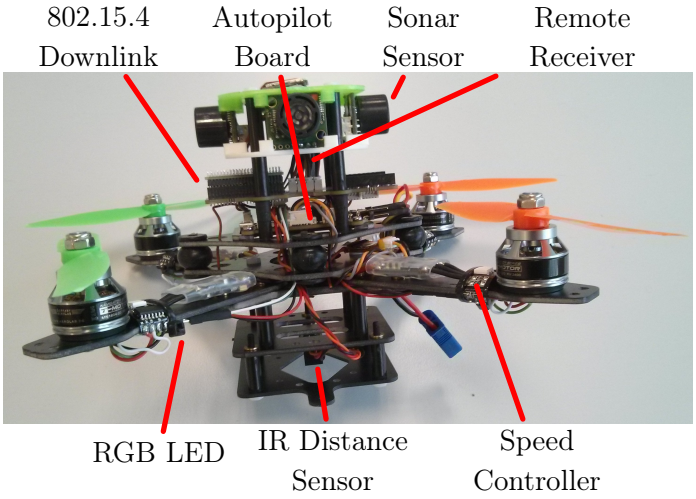


Figure 2.1: The components of a single FINken-III copter, described by [Steup et al., 2016].

This type of quadcopters typically subject to some constraints, which are explained in the following. The design of the platform is a challenge for itself and often discussed in literature [Bermes, 2010, Bouabdallah, 2007, Mettler, 2001, Pines and Bohorquez, 2006, Kumar and Michael, 2012]. Due to the small construction of our copters the load-capacity is limited. Thus, even restrictions apply on power storage, sensor payload and the flight times. By virtue of

[1]The FINken platform is developed at the SwarmLab of the Otto von Guericke University of Magdeburg, Germany (www.is.ovgu.de).

these limitations an energy-saving movement is even more important. In order to ensure the independence of external devices and sensors, the copters are equipped with multiple sensors. This includes four ultrasound distance sensors and one IR-distance sensor. Using these sensors, it is possible to autonomously hold the height and to recognize other copter or walls. Other projects [Kushleyev et al., 2013] that want to optimize the weight and size of the copters are dependent on external sensors.

The copter work with a customized version of the Paparazzi autopilot framework [Hattenberger et al., 2014]. The programming has been adapted that the copter can work without GPS. As a replacement, the distance and height sensors described above are used.

To model a swarm behavior the copters use an almost linear attraction and non-linear repulsion potential function between each other [Gazi and Passino, 2011]. The attraction and repulsion method is described in detail in [Steup et al., 2016].

The current version of the quadcopters uses a Li-Po Battery (3 Cells, 900 mAh). This battery is removable and must be changed manually. The battery allows a flight time of about 10 minutes.

For our proposed model an automatic loading would be desirable. There are several possibilities. The first idea involves charging via solar cells. Here the problem is again the weight. To guarantee a quick recharge, a lot of cells need to be installed. In addition, one has to rely on a source of light. A second possibility is described in [Mulgaonkar and Kumar, 2014]. Here the quadcopters fly autonomously for charging at predetermined stations. Furthermore, the possibility is introduced to transport the charging stations with UGVs (cf. Figure 2.2). With a combination of fixed and mobile charging stations a large area, like an earthquake-damaged building [Michael et al., 2012], can be explored.

Figure 2.2: The Pelican MAV being transported by the iRobot PackBot, described by [Mulgaonkar and Kumar, 2014].

3 Proposed Model

Our new approach consists of several parts, that are explained in detail in this chapter. The chapter is structured as follows. In Section 3.1, we describe our movement model for the individuals with the different states and actions. In order to employ PSO as a search mechanism for aerial micro-robots, we need to extend the PSO by a new energy model and additionally address the discrete time movements. For that, the discrete time flight concept is described in Section 3.2. Section 3.3 describes the neighborhood topology for the decision making process. Next, we describe the new simplified energy consumption model, based on the FINken-III copter, in Section 3.4. The model is developed for our FINken-III copter, but can easily be adapted to other robots. In Section 3.5, we show the algorithm of our new Energy Aware PSO approach and describe how the algorithm works. Afterwards, Section 3.6 describes the selection and multi-criteria decision making process, used in the EAPSO approach. Finally, we show the profit approximation for the decision making process in Section 3.7. The approximation is the main challenge in the decision making process. For that, we discuss some challenges and problems of the approximation method.

3.1 Movement Model

In order to better capture the main concept, we model the flight behavior of the individuals using a simple finite state machine, as shown in Figure 3.1.

The individuals operate in two states: “Ground” or “Air”. In the Ground state, the individuals can decide between two different actions. If an individual i has enough energy e_i (i.e., $e_i > e_{takeoff}$), it can start and switch to the Air state. In the case of low energy level ($e_i < e_{takeoff}$), the individual stays in the Ground state. For simplicity, we assume that staying in Ground state

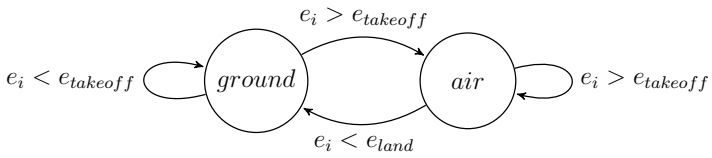


Figure 3.1: Transition graph depicting the flight model for an individual i depending on its energy level e_i

is coupled with charging. This is a very strong assumption which cannot be easily implemented in real-scenarios.

In the Air state the individuals either change state and go to Ground state if $e_i < e_{land}$ or they move (fly) towards a certain direction. In fact the direction towards which the individuals move has a great impact on the energy consumption of the individuals. In this case, the individuals need to decide on the distance they will move in the next iteration.

3.2 Discrete Time Flight

Even though the real-world is continuous in time, robotic systems work in a time-discrete way. They follow a strict input, compute, output scheme with a fixed time interval. In the case of real aerial micro-robots (quadcopters), different functional blocks have different time intervals. Our reference copters "FINken-III" [Steup et al., 2016] for example stabilize their attitude approximately 100 times per second, but they only decide about the next movement command approximately 10 times per second. Consequently, our simulation follows the time-discrete approach and models the micro-robots to only take decisions in certain time intervals. Each time interval is a snapshot of the system on which the next decisions will be taken. To generalize the simulation we abstracted away the time interval and simply count the amount of decisions taken, which also yields a monotonic increasing time measurement. This allows for an easier comparison with PSO Equations (2.1) and (2.2), since the equations are measured regarding the number of iterations passed. Therefore, we call our time interval count iterations.

3.3 Neighborhood Topology

In the following, we introduce the neighborhood topology for the individual. The neighborhood topology is required for the decision making process. The neighborhood includes the individuals who are necessary for the selection of the global best individual in the PSO formula. In the following, we take the "k-Nearest-Neighbor" neighborhood topology, where $k = N - 1$ refers to the fully connected network. In the experiments, the size k will be analyzed. An individual with a "k-Nearest-Neighbor" neighborhood communicates with his k nearest neighbors. It follows that each individual can only receive information from his neighborhood, e.g. the global best position or previously visited positions and the associated function values.

Choosing the size of the neighborhood depends on the risk value, described in Section 3.6. Also, the exchange of already known points in the landscape depends on the size of the neighborhood as described in 3.7. Other neighborhood topologies are also possible, but they are not examined in this thesis.

3.4 Energy Computations

In the following, we introduce a simplified model of energy consumption for the individuals which is modeled based on a real scenario from [Steup et al., 2016]. The energy of an individual is defined in units per iteration. In each iteration, each individual can use at most one energy unit. Based on this energy unit several constants are defined, which represent the energy consumption of different actions the aerial robot can execute, such as: take off, hover, control, move, and land. Each constant is defined as a fraction of an energy unit. The resulting constants are $e_{takeoff}$, e_{hover}, $e_{control}$, e_{move}, e_{land}, and e_{charge}.

$e_{takeoff}$ and e_{land} require a relatively fixed amount of energy and can be calculated by considering the required energy related to the potential energy:

$$E_{pot} = m \cdot g \cdot h \tag{3.1}$$

where the mass (m), the gravitation (g) and the target height h are considered to be constant as we let the aerial robots move at the same height (either the target height while flying or zero when landed). Therefore, we can conclude that $e_{takeoff} = E_{pot}$ and $e_{land} = -E_{pot}$.

$e_{control}$ indicates the amount of energy which is required for computations and communication and can be estimated to be a constant value per iteration for computation.

The main sources of energy consumption which can be additionally influenced by the individual itself, are the e_{hover} and e_{move}. The more time an individual spends in the hovering state, it will consume more energy. This also holds for the flight distance and its corresponding energy consumption denoted by e_{move}.

In order to keep the model as simple as possible, we have met the following assumptions. In our model, we set a maximum distance that each individual can move in one iteration. All robots fly at the same height, have identical weight and move in constant time steps. To calculate a simplified movement cost, the flight has an acceleration and a deceleration phase. With these assumptions, the cost for "move" depends on the flight distance which is calculated using the Euclidean distance values. The individuals can recharge their batteries while in Ground state. The charging rate e_{charge} is defined in percentage of the energy unit.

3.5 Energy Aware PSO

Algorithm 1 illustrates the main building blocks of our proposed approach called Energy Aware PSO. In this algorithm, we consider the flight physics in the PSO computations. The goal is to define a population of individuals, which search for an optimal solution in a defined search space. The individuals have physical constraints in terms of their energy values. In this algorithm, the individuals can decide about the amount of their movement in the search space based on both their current energy level and the performance of the other individuals in their neighborhood.

The algorithm starts with initialization of the individuals at $t = 0$ using certain start positions $\vec{x_i}(t)$, initial velocity values $\vec{v}_i(t)$ and certain energy level $e_i(t)$ for individual i.

After the initialization, the individuals make a decision about their functionality and select a state either as "Air" or "Ground". The function "DecideState" considers the amount of the energy $e_i(t)$ available to individual i and in case

Algorithm 1: Energy-aware PSO

Input : N Individuals
t $\leftarrow 0$
Initialize the individuals
for $i = 1$ *to* N **do**
    $\vec{x}_i(t) \leftarrow$ StartPosition(S)
    $\vec{v}_i(t) \leftarrow 0$
    $e_i(t) \leftarrow$ Random(e_{min}, e_{max})
end
while *Stopping Criterion not fulfilled* **do**
    for $i = 1$ *to* N **do**
        $state_i(t) \leftarrow$ DecideState($\vec{x}_i(t)$, $e_i(t)$)
        if $state_i(t) =$ *Ground* **then**
            charge: $e_i(t) \leftarrow e_i(t) + c$
        end
        else
            $\vec{x}_g(t) \leftarrow$ LeaderSelection($state_i(t)$, $e_i(t)$)
            $\vec{v}_i(t+1) \leftarrow$ ComputeVelocity ($\vec{x}_i(t)$, $\vec{x}_g(t)$)
            $\vec{x}_i(t+1) \leftarrow$ UpdatePosition($\vec{v}_i(t)$, $\vec{x}_i(t)$)
            $e_i(t+1) \leftarrow$ ComputeEnergy($\vec{v}_i(t+1)$)
        end
    end
    $t \leftarrow t + 1$
end

there is a certain minimum value of e_{min}, the individual takes off. Otherwise, the individual's state ($state_i(t)$) remains in "Ground" and gets charged. Here we take a simple additive recharging mechanism with a constant value c.

In the case that the individual takes off, it needs to find the globally best individual (denoted at the "LeaderSelection") to be able to perform the PSO movement. However, this depends on the amount of available energy $e_i(t)$. The individuals with enough energy values can perform as in the standard PSO, while the others with low energy values can only perform a local search. This decision has a large impact on the convergence of the approach and will be studied in Section 3.6. The next steps after the leader selection mechanism are straight forward. Each individual computes its velocity vector and moves

accordingly using the PSO equations. In addition to this, if the calculated velocity is less than a certain minimum threshold, e.g. if the individual is stuck in a local optima, we use the so called turbulence factor: those individuals take a randomly generated velocity vector. Additionally, we assign a maximal velocity value V_{max} as a threshold and in case the velocity vector is larger than V_{max}, it will set to V_{max}. After each movement, the individuals compute their energy consumption in "ComputeEnergy". This process is performed iteratively until a stopping criterion is fulfilled. We set the maximum number of iterations (time) as the stopping criterion.

3.6 Selection and Multi-Criteria Decision Making

Leader selection mechanism (cf. Algorithm 1) contains the multi-criteria decision making process for each individual in the population. An individual i must select a leader according to several factors such as its energy level e_i, the amount of overall work to be done and the status of the other individuals in the neighborhood. The main steps for selecting the leaders are shown in Algorithm 2.

Algorithm 2: Leader Selection for individual i

Input : $state_i(t)$ and $e_i(t)$
Output : Global best position $\vec{x}_g(t)$
if $state_i(t)$ = *Air* **then**
  for k = *1 to N-1* **do**
    $\vec{x}_g^k(t) \leftarrow$ FindBest(k)
    Cost(k) := ComputeCost($\vec{x}_g^k(t)$)
    Profit(k) := ComputeProfit($\vec{x}_g^k(t)$)
  end
  $x_g(t) \leftarrow$ MCDM(Profit, Cost, $e_i(t)$)
end

The individual i goes through the multi-criteria decision making for leader selection mechanism only when it is in the "Air" state. The first step is to find the globally best solutions for several neighborhood topologies with k =

$1, \cdots, N-1$ using the "k-Nearest-Neighbor". In this case, we can have $N-1$ different possible globally best solutions: $x_g^1(t)$ to $x_g^{N-1}(t)$. The individual computes (simulates) cf. 3.7 its next position $\vec{x}_i(t+1)$ by considering each of the possible $N-1$ globally best solutions. In order to select one of them, it computes the "cost" and "profit" in terms of energy consumption for each of the possible next positions. Cost simply captures the amount of required energy to reach $\vec{x}_i(t+1)$.

Profit means the difference between the quality of the current position $f(\vec{x}_i(t))$ and the next one $f(\vec{x}_i(t+1))$. This value must be approximated as the PSO equations involve several random values such as ϕ_1 and ϕ_2 and the quality of the position for $t+1$ is not known. Section 3.7 describes the approximation.

The values related to cost and profit are in conflict with each other; the solutions with high profit can cause a large energy consumption. In this case, the individual must select one of the $N-1$, $x_g^k(t)$, $k = 1, \cdots, N-1$, using concepts from multi-criteria decision making (denoted as "MCDM (profit, cost, $e_i(t)$)" in the Algorithm 2).

In this thesis, we take the weighted sum approach from Chapter 2. Each individual i is assigned a weight vector for the two criteria "profit" and "cost": $w_i = (r_i, 1-r_i)$, where r_i indicates the amount of *risk* in terms of energy consumption an individual would spend to achieve a large profit. For instance, $w_i = (1, 0)$ depicts the preference to select a new possible position which delivers a large amount of profit and requires a large amount of energy. The values for r_i can be selected using different mechanisms:

1. Randomly: Each of the individuals in the population has a random preference.
2. Constant: All the individuals have the same value such as (0.5, 0.5), (1, 0) and (0, 1).
3. Adaptive: The individuals select their preferences based on the amount of available energy.

After setting the preferences for the individuals, each individual ranks each of the possible $N-1$ new positions at $t+1$ according to its weight vector as follows:

$$Rank(k) \quad = \quad r_i \cdot profit(k) + (1 - r_i) \cdot cost(k) \tag{3.2}$$

Where $k = 1, \cdots, N-1$. The position with the lowest rank will be selected as the $\vec{x}_g(t)$ by the individual. In this case, the profit is a negative value, because we modeled the optimization as minimization problem.

The above multi-criteria decision making approach for each single individual implicitly implies that the individuals with low values of risk (e.g., $r_i = 0$) will perform small movements in the search space and hence a local search. On the other hand, the individuals with $r_i = 1$ select the leaders which are far away from them and require a large amount of energy. Considering the amount of profit in the decision making process implicitly involve the status of the other individuals in the neighborhood. If all of the individuals in a neighborhood have more or less the same function value, the amount of profit will degrade.

3.7 Profit Approximation

In this section, we describe the approximation of the profit by the individuals. As described in the last section, each individual simulates the next $N-1$ possible steps in order to make a decision. The output of this process is a set of parameters. This set contains the next state of the individual, the action performed by the individual, the cost for moving, the velocity vector and the profit of the movement. Since we deal with an unknown environment, the function value of a none visited position is not known. Therefore, the function value of the possible next positions must be approximated. Here, we use the information given by the neighborhood around each individual who is able to access the previously visited points and corresponding function values of all individuals in its neighborhood. With this information, the individual is able to approximate the unknown landscape and can calculate the function value of the simulated goal. For the approximation, we use ordinary least-squares regression [Moutinho and Hutcheson, 2011] to fit a quadratic model with constant, linear, interaction, and squared terms. In order to save memory and improve the approximation, each individual collects points with a distance greater than a certain threshold (here 0.1) in the neighborhood.

Challenges and Problems

In this section, we describe the challenges and problems of the approximation method in more detail.

The main challenge of the approximation is that we only know already visited points and not the entire landscape. A second challenge is that in most of the cases the individuals just fly a short distance and the information gain of the new point is very low. The third challenge is the size of the approximation. If we approximate the entire landscape, we consume unnecessary resources like energy and computation time. As well, the approximation of the entire landscape with only limited given points is not sufficient.

Figure 3.2 shows the approximation of the Rosenbrock function after 500 iterations for one individual (as full landscape approximation). One can see that the Rosenbrock function is not completely approximated. This is due to the fact that a large part of the landscape is not known after the 500 iterations. Therefore, only a small part of the landscape is known. The approximation from Figure 3.2 represents the valley of the function.

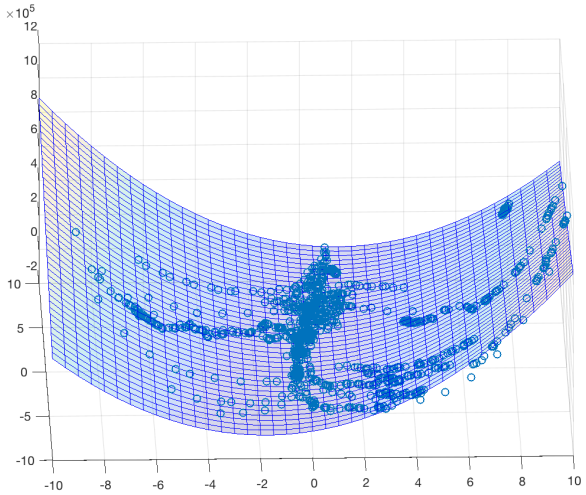


Figure 3.2: Approximation of Rosenbrock function after 500 iterations with entire landscape

Figure 3.3 shows a close up view from Figure 3.2. Here it becomes clear how much the approximation deviates from the given points. The known points are all close to zero. The approximated surface is near $-2 \cdot 10^4$ and thus shows a significant deviation. With the poorly distributed given points it is difficult, with a fairly simple method, to approximate the landscape.

To avoid the problems, we use some techniques, we describe in the following. As describes in Section 3.7, we only add points with distance greater than 0.1

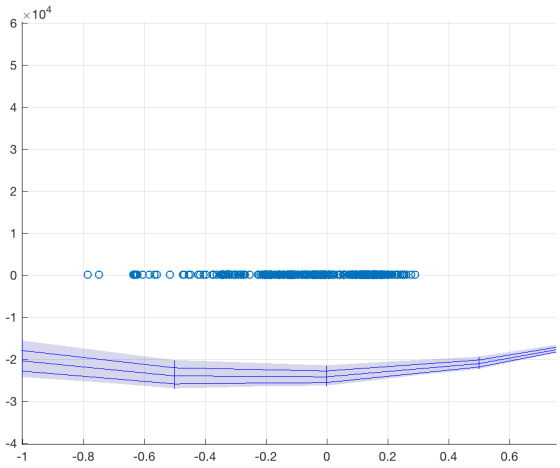


Figure 3.3: Clipping of Rosenbrock approximation (Figure 3.2)

to the existing ones. This saves memory, speeds up the calculation and improves the approximation. As second improvement, we only use the k nearest neighbors of the point we want to estimate. The approximation of the neighborhood is faster and more precise. Test shows that $k = 20$ gives the best results for all functions. Another reason why we use the neighborhood approximation is, each individual can only move a limited distance per iteration, so the hole landscape is not important for the decision making. The results show that the neighborhood approximation is better than the approximation of the hole landscape.

Figure 3.4 shows the average error of the approximation per iteration for the three functions. The plot shows 30 runs of our adaptive EAPSO method. The peaks in the curves are caused by the collective fly and load behavior of the individuals. The approximation error plot for the Sphere function displays up to iteration 200 an error. The error disappears when the minimum is reached. In the beginning, only a part of the function is known, and this is approximated as a surface. The first peak is caused by the lack of data points. In the first iteration, the individual knows only his own position, so the corresponding error is high.

The error plot for the Ackely function shows a periodically uniform approximation error. The error remains until the end. Here, the error can be explained by the local optima. The method only approximates the local optima. Thus,

an accurate calculation of profit is very difficult. The shown error is lower than in the approximation of the whole landscape.

The Rosenbrock function shows the greatest approximation error. The rash at the beginning can be explained by the small number of data points (similar to the other functions) and the high starting positions of some individuals. The error decreases when the valley is reached. After reaching the minimum, the error is reduced even further.

All three functions show an approximation error, but the error is smaller than using the approximation of the whole landscape. Only for the Sphere function, the error drops to zero in the time of recharging. This is due to the landscape of the functions. Individuals move uniformly, but not all at the same time. The Sphere function is very simple, therefor, we can get a very uniform motion. For functions with local optima, it is more difficult. No steady motion can take place here. As a result, there are always individuals who move. Thus, the approximation never drops to zero.

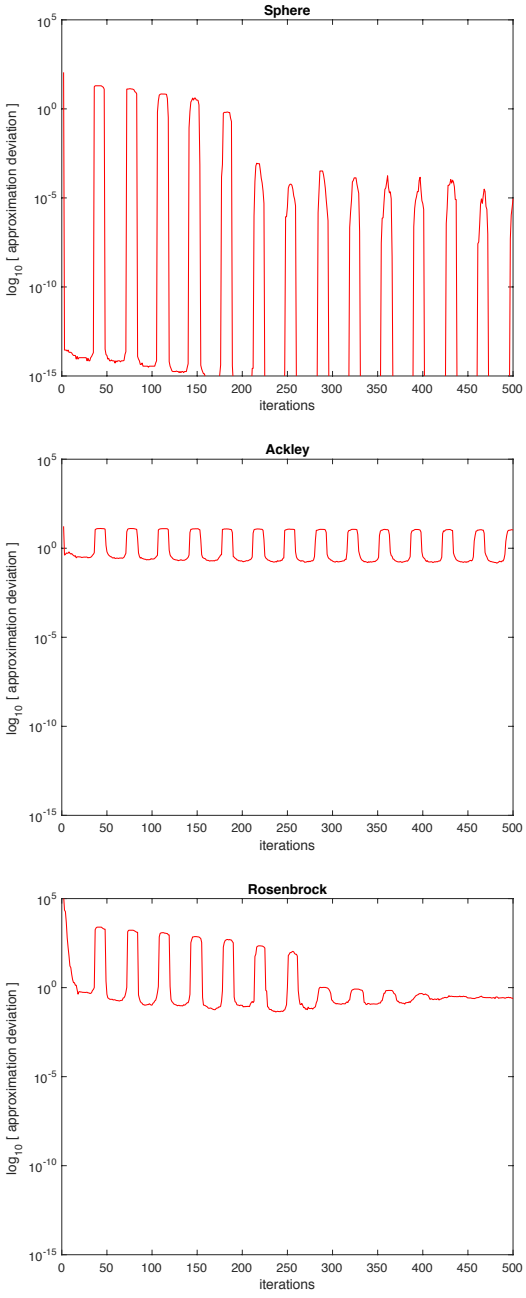


Figure 3.4: Average approximation deviation for the three functions over iterations

4 Implementation

This chapter describes the implementation of the proposed EAPSO model from Chapter 3. First, Section 4.1 describes the implementation of the swarm component of the PSO algorithm and Section 4.2 the corresponding particle or individual. In Section 4.3, we describe the implementation of the neighborhood topology, introduced in Section 3.3. Afterwards, Section 4.4 shows the implementation of the state (described in Section 3.1), Section 4.5 explains the implementation of our simplified energy model (described in Section 3.4) and Section 4.6 the selection and multi-criteria decision making process from Section 3.6.

To make the implementation extensible and usable for future reference, an object-oriented programming was chosen for the project. The project is implemented in MATLAB. The used MATLAB version is R2016a. The code can be downloaded from `https://github.com/wittfabian?tab=repositories`.

4.1 Swarm

As the central control unit operates the swarm class. In the class, the initialization and the central control of the individuals is implemented. The most important part at it is to control the optimization using the "optimize" method, described in Algorithm 3. Here, the command of the optimization is forwarded via the method "move" (cf. Algorithm 4) to the individuals. In addition, the method stores all important information such as battery levels, positions, function values, the global best individual and the neighborhood topology of each individual. The stopping criterion here can be either the number of iterations, a minimum of the function value or a threshold of change in the function value.

For the optimization process, the "move" method delegates via "simulateMove" and "move" the movement command to the individuals. This is recommended,

Algorithm 3: Global optimization (*swarm/optimize*)

```
Input : Stopping Criterion
while Stopping Criterion not fulfilled do
    move()
    saveEnergyLevels()
    saveFunctionValues()
    saveGlobalBestIndividual()
    saveNeighborhoodTopology()
end
```

Algorithm 4: Delegation of the move command (*swarm/move*)

```
Input : N Individuals
for i = 1 to N do
    simulateMove(individual i)
end
for i = 1 to N do
    move(individual i)
end
```

because first, all individuals have to simulate the move, including the decision making process for the neighborhood topology. After that, they can execute the move. Only with this order, we can simulate a parallel decision making and movement process.

4.2 Particle

The particle class describes the individual with his properties like position, function value, energy (as object, described in Section 4.5), neighborhood (as object, described in Section 4.3), state (as object, described in Section 4.4), risk value, landscape memory, option whether constant ϕ_1 and ϕ_2 is used or not, option whether the turbulence factor is enabled and the task. The particle class also contains the PSO values ϕ_1, ϕ_2, C_1, C_2, w and the last velocity $v_i(t)$.

The task is implemented as a class and currently only includes the optimization function. To facilitate the further work on the project and to allow multi task

scenarios, the tasks functionality was implemented as class. The risk value is dependent on the scenario, and either determined during the initialization (random or pre-defined) or is dependent on the current energy. The option whether ϕ_1 and ϕ_2 is constant or not affected the generation of ϕ_1 and ϕ_2. If the option is enabled, the PSO algorithm uses for each simulation in the actual iteration the same ϕ_1 and ϕ_2. Otherwise, the variables are generated before each simulation is performed. In addition, the movement of the individual is stored as a pair of position and functional value per iteration (named *movement_memory*). Also, the deviation between the approximate function value and the real function value is stored, to possibly serve for a learning process (named *save_approximation_values*).

To allow the individual to decide between different movements, we simulate the PSO movement with different neighborhoods (cf. Algorithm 5) and save everything in the so called simulation container. The simulation container is used as input for the decision making process and saves the state, action, needed energy, velocity vector and the approximated profit. The set of possible actions contains: start, land, hover, fly, wait and charge.

Algorithm 5: Decision making process (*particle/makeDecision*)

Input : Individual I
Output : best individual
for *k = 1 to N - 1* **do**
    nh_knn(k) ← `copy(`*Individual I*`)`
    nh_knn(k).`neighborhood` ← `neighborhoodKNN(`*k*`)`
    nh_knn(k).`simulateMove()`
end
Rank ← `LeaderSelection(`*risk selection*`)`
Rank.`addParticleArray(`*nh_knn*`)`
Rank.riskval ← *risk value of individual I*
best Individual ← Rank.`run()`

The method "makeDecision" generates for each neighborhood size a copy of the individual and simulates the move with this neighborhood. After that, all simulations are added to the ranking object and ranked with respect to the risk value. The function "run" performs the selection mechanism, including the dimension generation, ranking, and selection, and returns the best individual.

4.3 Neighborhood Topology

The neighborhood class provides, dependent on the size, all relevant informations. The most important information is the global best individual, as well as its position and the corresponding function value. The position and the function value of the global best individual is also stored in the particle class. The implementation currently uses only the "k-Nearest-Neighbor" topology. For the purpose of reusability and extensibility, the class is modeled as abstract to possibly test and use new neighborhood topologies. The neighborhood object knows all individuals in the swarm and individually selects the k nearest. Each neighborhood object has to implement the "getGlobalBest" function. The function returns the global best individual dependent on the chosen value k. Another function of the neighborhood is the collecting of visited positions. The "getParticleLandscapeMemory" function collects all visited locations, with the corresponding function values, of the neighbors and returns them as a list.

4.4 State

The state class is the core of the movement model. The state class is also modeled as an abstract class, to simply add new states. As defined in Section 3.1, we decide between two sates: "Ground" and "Air". Each state has to implement the two functions "simulateMove" and "move" (used in Algorithm 4). The action property describes the actual behavior in the state. In the Ground state the individual can decide between the action "charge" or "wait". In our approach, the action "wait" is only performed in the first iteration when the individual wait to start. In all other cases the individual automatically switches to the charging mode. Algorithm 6 describes the method "simulateMove" as implementation in the state ground class. If $energy \geq e_{takeoff}$ is true, the individual is able to start. In this case, we calculate the necessary energy for the start (named $energy\_change$) and store all relevant parameters in the simulation container with the help of the function "setSaveSimulate". Otherwise, if the energy of the individual is too low, the individual stays in the Ground state and switch to the recharging process.

Algorithm 7 describes the method "simulateMove" as implementation of the state air class. The operation of the method is similar to the method in the state ground. The method distinguishes three cases:

Algorithm 6: Movement simulation (*Ground/simulateMove*)

if $energy \geq e_{takeoff}$ **then**
    energy_change ← `calculateEnergyCost(`*start*`)`
    `setSaveSimulate(`*state air, action start, energy_change,* $\overrightarrow{0}$*, 0*`)`
    return
end
`setSaveSimulate(`*state ground, action charge, 0,* $\overrightarrow{0}$*, 0*`)`

- If the energy is less than e_{fly}, the individual switch to the Ground state and start recharging
- If the energy is greater or equal than e_{fly}, the individual is able to simulate the fly (cf. Algorithm 8)
- If none of the first two options are possible, the individual switches into the hovering mode.

The last option is not possible (with this implementation), but is added for the sake of completeness.

Algorithm 7: Movement simulation (*Air/simulateMove*)

Input : Individual P
if $energy < e_{fly}$ **then**
    energy_change ← `calculateEnergyCost(`*land*`)`
    `setSaveSimulate(`*state ground, action land, energy_change,* $\overrightarrow{0}$*, 0*`)`
    return
end
if $energy \geq e_{fly}$ **then**
    `simulateFly()`
    return
end
energy_change ← `calculateEnergyCost(`*hover*`)`
`setSaveSimulate(`*state air, action hover, energy_change,* $\overrightarrow{0}$*, 0*`)`

Hereinafter, the operation of the method "simulateFly" will be explained. The method includes the calculation of the PSO formula and possible adjustments. First, the vector $v_i(t+1)$ is calculated by using the PSO formula from Section

2.1. Next, we check whether the condition for the turbulence factor is fulfilled. The condition is true if the length of the vector $v_i(t+1)$ is smaller than 0.001. In the next step, we check whether the length of the vector exceeds the maximum length V_{max} (defined in Section 3.4). If the statement is true, the vector is shortened via the function "shortVector". For the obtained velocity vector the resulting energy cost and new energy level of the individual is calculated by using the function "calculateFlyCost". If the energy value, after deduction of the flight costs, is lower than the threshold e_{fly}, the fly is shortened again. For this, the usable energy is calculated. If the usable energy is less or equal than zero, we return without a fly. Otherwise, we shorten the vector $v_i(t+1)$ with the function "shortVector" with respect to the usable energy. As the last step, the profit of the fly is approximated with "calculateProfit" (described in Section 3.7). In the end, the simulated parameters are saved via "setSaveSimulate" in the simulation container.

Algorithm 8: Flight simulation ($Air/simulateFly$)

Input : Individual P

$\vec{v}_i(t+1) \leftarrow \omega \cdot \vec{v}_i(t) + \phi_1 \cdot C_1 \cdot (\vec{P}_{best}(t) - \vec{x}_i(t)) + \phi_2 \cdot C_2 \cdot (\vec{x}_g - \vec{x}_i(t))$

if $\|\vec{v}_i(t+1)\| < 0.001$ **then**
  $\vec{v}_i(t+1) \leftarrow$ `addTurbulenceFactor(`$\vec{v}_i(t+1)$`)`
end

if $\|\vec{v}_i(t+1)\| \leq V_{max}$ **then**
  `// short` $\vec{v}_i(t+1)$ `to` V_{max}
  $\vec{v}_i(t+1) \leftarrow$ `shortVector(`$\vec{v}_i(t+1)$, V_{max}`)`
end

energy_change, new_energy ← `calculateFlyCost` $(\vec{v}_i(t+1))$;

if $new\_energy \leq e_{fly}$ **then**

  $usableEnergy \leftarrow e_i(t)$ - e_{fly}

  if $usableEnergy \leq 0.0$ **then**
    return; `// return without fly`
  end

  `// short` $\vec{v}_i(t+1)$ `with` $usableEnergy$
  $\vec{v}_i(t+1) \leftarrow$ `shortVector(`$\vec{v}_i(t+1)$, $usableEnergy$`)`
  energy_change ← `calculateFlyCost` $(\vec{v}_i(t+1))$;
end

profit ← `calculateProfit(`$\vec{v}_i(t+1)$`)`;
`setSaveSimulate(`*state air, action fly, energy_ change,* $\vec{v}_i(t+1)$, *profit*`)`;

4.5 Energy

The energy class implements our simple energy concept from Section 3.4. Here, the model can be adapted to the special properties of each copter. The class holds the actual energy level of the individual, the definitions of the energy values (described in Section 3.4, e.g. $e_{takeoff}$ or e_{hover}), the maximum movable distance per iteration V_{max}, and is able to calculate the energy consumption for a given movement (e.g. start, land, hover or fly). The functions "doAction" and "calculateDoAction" contain the rules for calculating the possible actions. The method "calculateDoAction" only calculates the energy consumption and the new energy level for a given action. The method "doAction" performs the action, which means that the function changes the actual energy level of the individual.

4.6 Leader Selection

The selection of the best simulation is performed in the Leader Selection class. Algorithm 9 describes the Leader Selection mechanism from Section 3.6. The algorithm receives as input the simulated individuals with the corresponding simulation containers and the risk value R. In the first step, the data is extracted from the container and stored in a dimension table. It should be noted that the profit is stored as negated value because we modeled the optimization as minimization problem. As the second step, we normalize the dimensions. Afterwards, the rank of each element or dimension is calculated, described in Section 3.6. In the final step, the smallest element of the ranking list is returned by the function "min". If several elements have the lowest rank, one element is randomly selected.

Algorithm 9: The Leader Selection algorithm

Input : N Individuals, risk R
for *i = 1 to N* **do**
    simCont ← `simulationContainer(`*i*`)`
    dimensions.`add(`*simCont.cost, simCont.profit * -1*`)`
end
dimensions ← `normalize(`*dimensions*`)`
`// calculate risk ranks`
for *i = 1 to N* **do**
    rank(*i*) ← *dimensions*.profit * *R* + (1 - *R*) * *dimensions*.cost
end
`// get best ranked element`
best ← `min` (*rank*)
if `size(`*best*`)` > *1* **then**
    return *best*.random
end
else
    return *best*
end

5 Evaluation

This chapter presents the experimental evaluation of EAPSO. The individual EAPSO methods are shown and compared. How the concepts work will be shown along with the impact of different fitness functions in comparative experiments. The following experiments are performed using MATLAB R2016a.

5.1 Experiments

The main idea of this thesis is motivated by a real case scenario of an aerial micro-robotic swarm. The proposed model and the corresponding features are meant to provide an algorithmic design for the energy consumption and search of the aerial swarm. Therefore, the goal of the experiments is to provide a baseline for further realistic tests. The parameters are selected based on a model of the FINken-III micro aerial robot [Steup et al., 2016], described in Table 5.1.

The flight of the aerial robots is modeled in an $n = 2$ dimensional search space. The goal is to analyze if a swarm with limited energy can find an optimal solution in the search space. We take the standard PSO [Kennedy and Eberhart, 2001] as the baseline algorithm and denote it as "default PSO". As the search space only contains two parameters, the default PSO is easily able to solve the problem.

Three test problems such as Sphere, Ackley, and Rosenbrock from the literature are being used for the experiments. These test problems can very well simulate the terrain in which aerial swarms can fly and search, while Sphere is only for simple tests, Ackely and Rosenbrock respectively capture search terrains with lots of local optima and a flat plateau. The primary focus of our experiments is the different multi-criteria decision making approaches and the total energy consumption. All the experiments run 30 times. The median values and corresponding standard errors are reported.

The individuals fly in a defined arena. All individuals start in a defined area with a random position (this is a realistic assumption for aerial robotic systems). The initial state is set to be the Ground state. The optimization stops after 500 iterations. All parameters are listed in Table 5.1.

Table 5.1: Selected parameters for the experiments

Description	**Parameter**	**Value**
PSO variables		
population size	N	30
initial velocity	v_0	0.0
inertia weight	w	0.5
attraction rate for $\vec{P}_{best}$	C_1	1.0
attraction rate for $\vec{x}_g$	C_2	1.0
random value for $\vec{P}_{best}$	ϕ_1	random
random value for $\vec{x}_g$	ϕ_2	random
energy consumption		
takeoff	$e_{takeoff}$	10
hover	e_{hover}	28
control	$e_{control}$	20
move	e_{move}	22
land	e_{land}	-10
charge	e_{land}	10
flight area		
search space	n	2
area length	x_1	$[-10, 10]$
area width	x_2	$[-10, 10]$
start area length	x_{S1}	$[-10, 10]$
start area width	x_{S2}	$[-10, -8]$

In all experiments, the best function values (denoted as "fitness"), the total amount of movement in the swarm (denoted as "distance") and the total amount of available energy in the swarm (denoted as "energy") are measured. The total amount of movement is meant to capture the amount of energy consumption, while the total amount of energy can be used to estimate the

charging behavior and its frequency during the 500 iterations. In the experiments, we compare default PSO with EAPSO with different values for r_i as 1.0 (named *risky*), 0.5 (named *balanced*) and 0.0 (named *careful*). Additionally, we perform a random risk initialization of the individuals denoted as *random* and the adaptive variant denoted as *adaptive* in which the individuals select a leader according to their available energy level $e_i(t)$.

5.2 Simulation Environment

Figure 5.1 shows the simulation environment (Rosenbrock function) at the first iteration. Here one can see that all individuals start in the specified area from Table 5.1. The position is random. The color of the landscape denotes the function value. Bright colors stand for high function values and dark colors for lower functional values. The valley can be seen clearly.

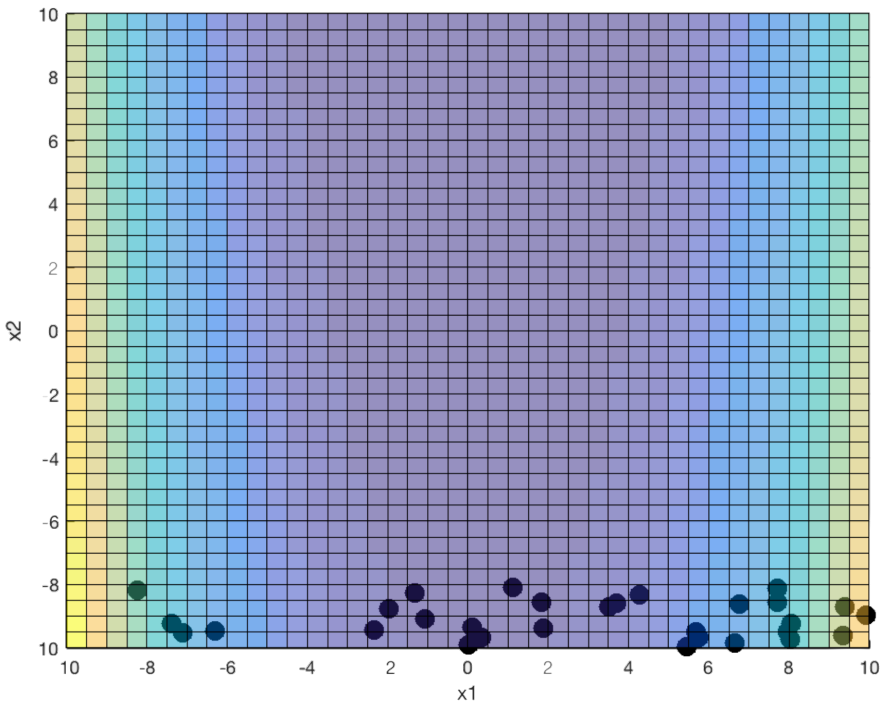


Figure 5.1: Simulation environment for the Rosenbrock function with our adaptive EAPSO concept at the first iteration and 30 individuals. The black dots represent the individuals at their current position

Figure 5.2 shows the simulation environment (Rosenbrock function) after 30 iterations. Most of the individuals are close together. Some of the points are colored red. This means that the individual is currently recharging.

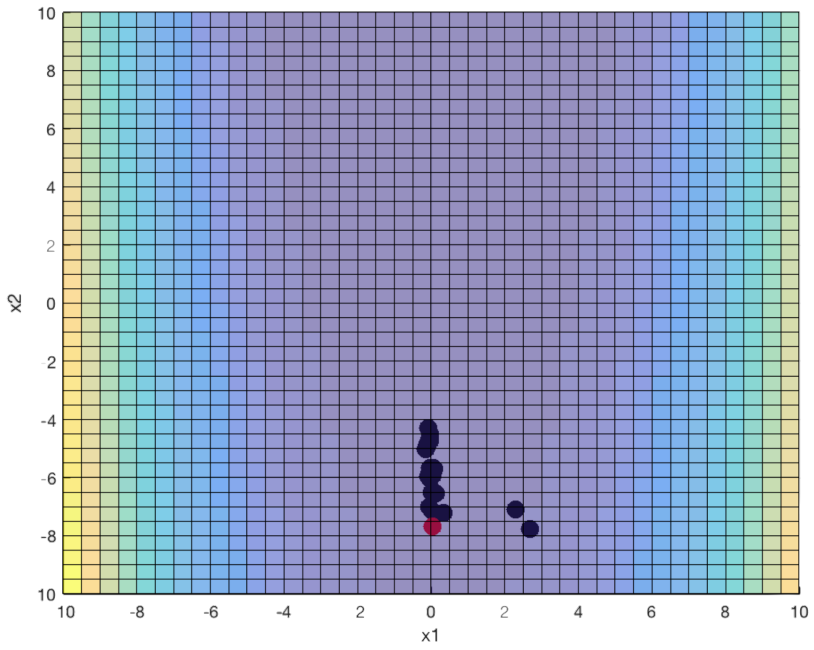


Figure 5.2: Simulation environment for the Rosenbrock function with our adaptive EAPSO concept after 30 iterations and 30 individuals. The black dots represent the individuals at their current position. Red dots indicate charging individuals.

5.3 Results

In order to better analyze the results, we investigate the convergence plots, the energy and distance measures over the iterations.

The detailed analysis is structured as follows. First, we examine all three plots (convergence, energy and distance) for the constant risk types $r_i = 0.0$ (named $careful$), $r_i = 0.5$ (named *balanced*) and $r_i = 1.0$ (named $risky$) in Section 5.3.1. The consideration is to show how the decision factors (movement cost and profit) affects in general. Then, in Section 5.3.2, we compare the two types *balanced* and *random*. For the risk type *balanced*, all individuals of the swarm have the same level of risk. In the case of *random*, the value is determined randomly in the initialization of the individual. The value can vary from individual to individual. Here the difference between a homogeneous swarm and a heterogeneous swarm (with different risk values) can be examined. Next is the comparison of *random* and *adaptive* in Section 5.3.3. Both methods have different characters in the swarm. In the case of *random*, the risk values are constant over time, however, *adaptive* calculates the risk value out of the energy level. Here the benefits of dynamic risk values are shown. Afterwards, we compare our *adaptive* EAPSO with the default PSO in Section 5.3.4. Here one can see the advantages and disadvantages of the new concept. Finally, we discuss some details and characteristics of the different types that are not apparent from the plots.

5.3.1 Constant Risk Analysis

In this section, we consider the constant risk types $careful$, *balanced* and $risky$. We show that the risk types work as desired and talk about their properties. First, we have a look at the convergence plot in Figure 5.4. The curves confirm the concept of the risk values. The higher the risk value is, the faster the optimum is reached (for this simple function). *Balanced* and $risky$ here differ only slightly. The largest deviation can be seen for $careful$. The method shows the worst result and does not reach the optimum (cf. Table 5.2).

In the next plot, one can see how the risk value affects the moved distance per iteration. For $careful$, an average movement of about 1.9 can be seen at the beginning. This value decreases very fast. This corresponds to the desired behavior with minimal movement. The larger movements at the beginning are caused by the distribution of the individuals. Once the individuals have collected, they move only minimally. The contrary form the individuals with the risk value $risky$. Here one can see, that at the beginning the maximum distance per iteration is selected. Again, the intended behavior is generated. The curve shows quite clearly when most of the individuals have reached the optimum. The individuals in this method decide solely on the basis of profit. After reaching the optimum, due to the shape of the Sphere function, the approximation returns in most of the cases a negative profit. Therefore, the individuals choose small neighborhoods and move only minimal distances. The third variant includes the individuals with the risk value *balanced*. These individuals weight the cost and profit of a movement with 50 percent each. The average distances here lie between the two other methods. Due to the short distances, these individuals take a little longer until they have found the optimum. Here can also be seen when most of the individuals have reached the optimum. In this case, the individuals behave similarly to the ones with the risk value $risky$. Also, the further course is very similar.

Next, we have a look at the average total amount of energy in the swarm per iteration. *Balanced* and $risky$ show a nearly identical course. Only the curve of $careful$ is shifted slightly to the right. The lower power consumption effects that the first charge cycle takes a little later. The uniform shape of the curves shows that the individuals are very in sync. If one compares the total amount of energy in Table 5.2, one arrives at the same conclusion. *Balanced* and $risky$ need almost the same amount of energy. Only $careful$ requires less but is not able to reach the optimum.

Table 5.2: Results of the Sphere function with risk *careful*, *balanced* and *risky* (median values and standard errors (std)) "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
careful	5.598	0.485	4375.483	1.499	415.959	5.337
balanced	0.000	0.000	4430.815	1.157	729.013	2.684
risky	0.000	0.000	4449.644	1.203	799.311	3.406

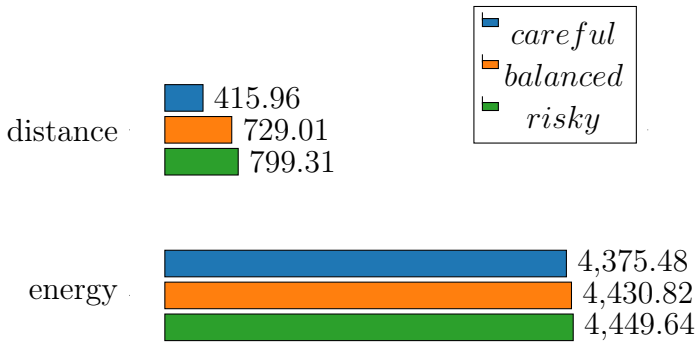


Figure 5.3: Results of the Sphere function with risk *careful*, *balanced* and *risky* (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

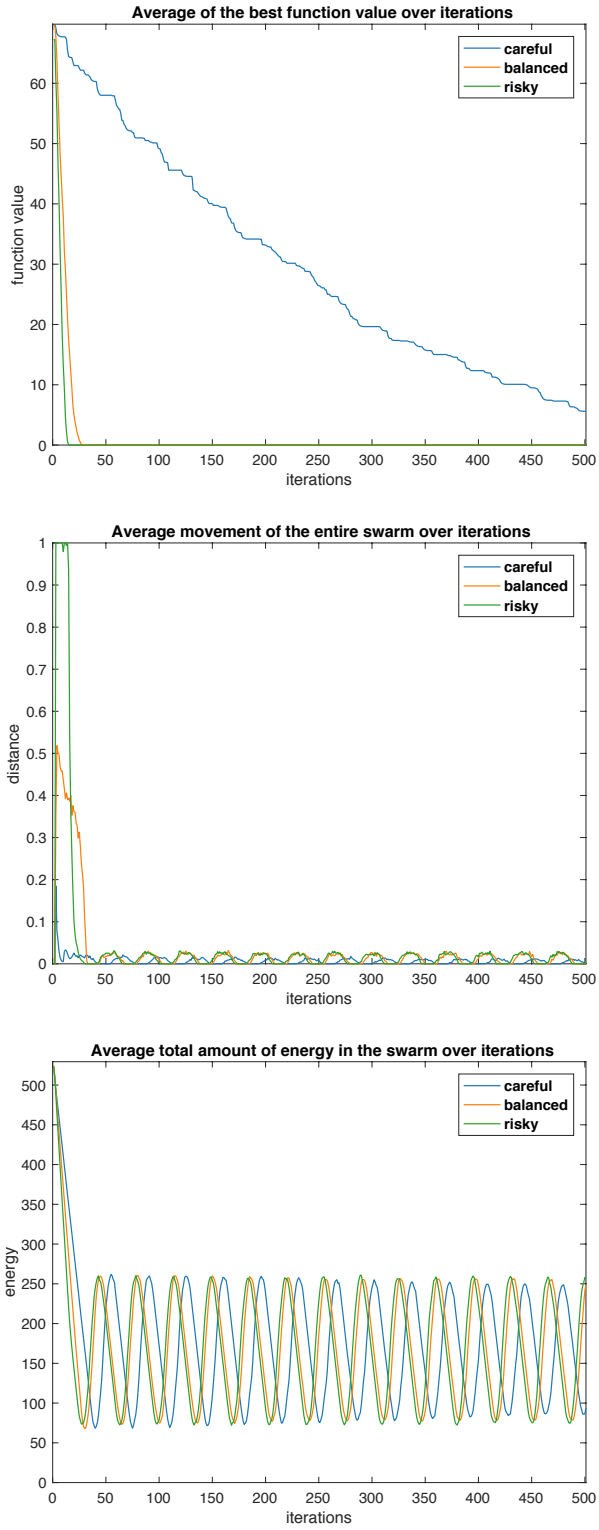


Figure 5.4: Convergence plot, energy plot and movement plot for Sphere function with risk *careful*, *balanced* and *risky*

Next, we consider the Ackley function in Figure 5.6. Again, the convergence plot shows a similar course as the Sphere function. *Balanced* and *risky* achieve the optimum in a short time. *Risky* needs a little less time. For this function, again *careful* does not reach the optimum (cf. Table 5.3). Only a minimal improvement takes place here. This is due to the landscape of the function. *Careful* always chooses the way with the lowest cost and thus the shortest path. This means that the global best is always very close. Due to the local optima of the Ackley function, it is therefore very difficult to leave the local optima. Overall, one can say that the risk types show here the expected behavior for the convergence plot.

Next, we have a look at the average movement per iteration. Here the curve for *risky* stands out particularly. We can see relatively large movements, although the optimum has been reached. Table 5.3 shows a multiple higher total moved distance for the *risky* method. This behavior is also caused by the local optima of the function. A few individuals have reached the optimum, but many individuals still stuck in local optima. The attempt to achieve the global optima produces the increased distances. The constant height of the average moved distances shows that most individuals have not reached the optimum. The other method behaves similarly. *Careful* and *balanced* show high average movements. From iteration 150, the average movement for *careful* is even higher than for *balanced*. *Careful* continuously tries to reach the global optimum and therefore the average movement is higher. Most individuals of the *balanced* method, however, have at iteration 150 already reached the optimum. Therefore, the average movement decreases continuously.

The third plot (average total amount of energy in the swarm) confirms the behavior of the methods for the function. The decreasing amplitude, in the energy plot, shows that the individuals are no longer in sync. This means the number of individuals who simultaneously charge or fly decrease over time. This behavior is as well caused by the local optima. Individuals at different positions require a different number of iterations, and therefore different amounts of energy to leave local optima. Referring additionally the convergence plot, it is striking that the optimum is reached relatively quickly (for *balanced* and *risky*), but the power curve does not change appreciably from the time of convergence. Some individuals can quickly reach the optimum. These individuals do not move noticeably and use the larger part of their energy to hover. Another part of the swarm stuck in local optima and thus requires more energy for movement. This means that the synchronization is

lost in the swarm. This course is shown by all three methods. Most clearly it is for the *risky* method.

The analysis shows that only the consideration of the profit for a function with local optima (e.g. Ackley function) is very inappropriate and consume unnecessary energy. The *balanced* method takes a little longer to reach the optimum but consumes almost 1.5 times less energy than the *risky* method. This shows the advantage of the multi-criteria decision making process.

Table 5.3: Results of the Ackley function with risk $careful$, *balanced* and $risky$ (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
careful	11.61	0.162	4505.048	3.383	935.639	9.089
balanced	0.000	0.554	4531.115	8.800	1008.798	22.184
risky	0.001	0.171	4993.991	6.649	2491.168	25.109

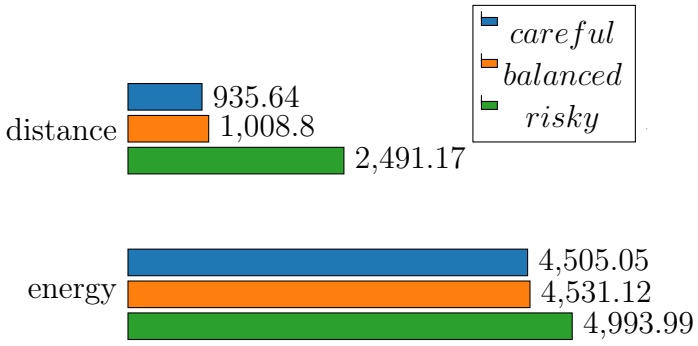


Figure 5.5: Results of the Ackley function with risk $careful$, *balanced* and $risky$ (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

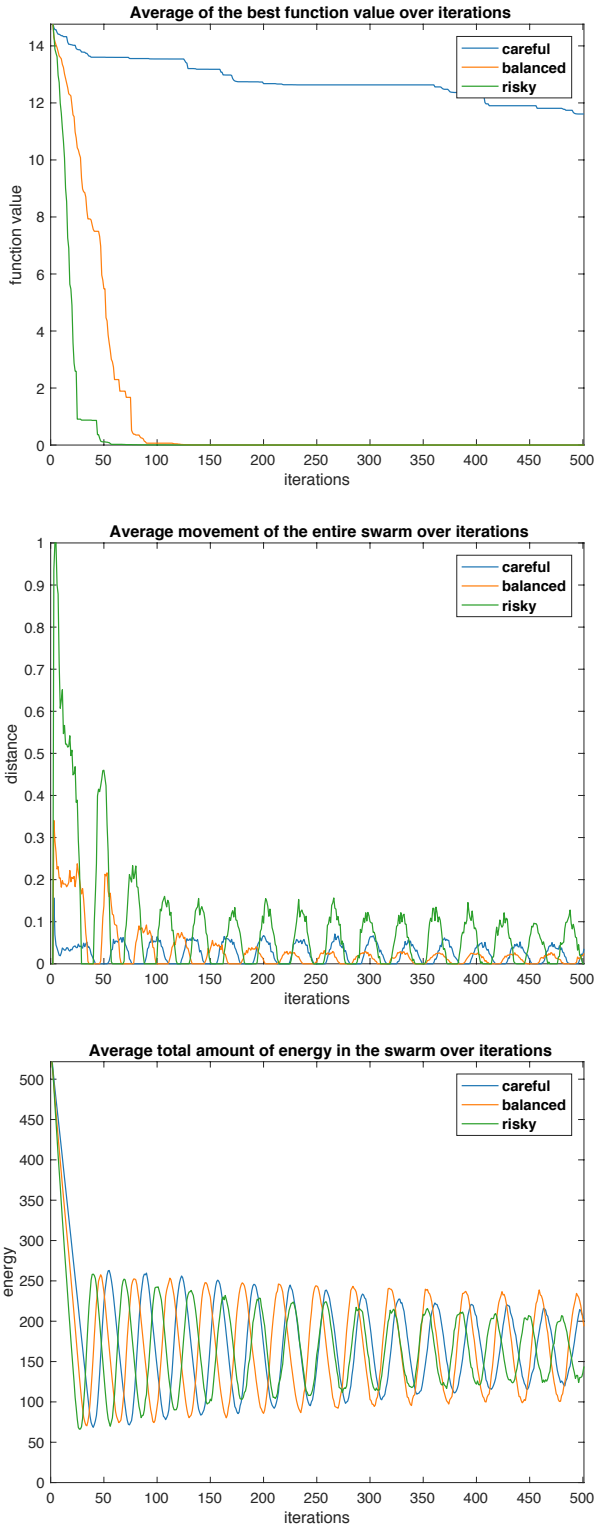


Figure 5.6: Convergence plot, energy plot and movement plot for Ackley function with risk $careful$, $balanced$ and $risky$

In last part of this Section, we have a look at the Rosenbrock function in Figure 5.8. Again, the convergence plot shows the differences of each method and the corresponding risk values. All three methods require more time, in order to achieve the optimum. The exception is here again *careful*, the method does not reach the optimum. Also, *balanced* and *risky* does not reach the optimal value 0.0 (cf. Table 5.4). Noticeable here is the standard error for *careful* and *balanced*. This indicates that some runs did not reach the optimum. This suggests that small risk values for flat landscapes are unsuitable.

Considering the average moved distance per iteration, *balanced* and *risky* show a different behavior. *Balanced* achieves the optimum only at iteration 350 and almost no movement takes place until this point. The curve is nearly like the *careful* method. *Risky* shows a relatively large movement, although the optimum is already reached at iteration 125. On the flat landscape of the function, the two methods respond very differently. Since *balanced* also includes the cost of the movement, the resulting movement is shorter. *Risky* in contrast only focuses on the profit. Here, it often happens that too large movements are selected and individuals wander. *Risky* shows here the same disadvantages as for the Ackley function in Figure 5.6. The average moved distance for *risky* is even 3 times as high as for *balanced* (cf. Table 5.4).

A view of the energy curves supports this theory. *Careful* and *balanced* show a consistent and similar course. The only change is the reduction in the amplitude over time. The reduction of the amplitude is caused by the desynchronization of the individuals. Conspicuous is this behavior in the *risky* method. Here the curve shows a zigzag course. The actions of the individuals are very different. Some individuals have already found the optimum and do not move. Other individuals are attracted by the swarm and maybe overfly the optimum. A third type breaks out from the swarm after recharging and leaves the optimum again. Thus, individuals are always in movement and energy is consumed.

Table 5.4: Results of the Rosenbrock function with risk *careful*, *balanced* and *risky* (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
careful	1095	76.57	4391.300	2.108	491.280	8.549
balanced	1.483	68.10	4432.276	7.299	714.238	28.774
risky	0.102	0.296	4899.941	17.155	2179.118	58.146

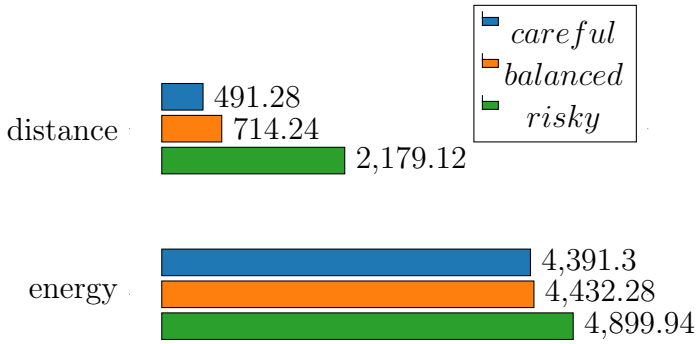


Figure 5.7: Results of the Rosenbrock function with risk *careful*, *balanced* and *risky* (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

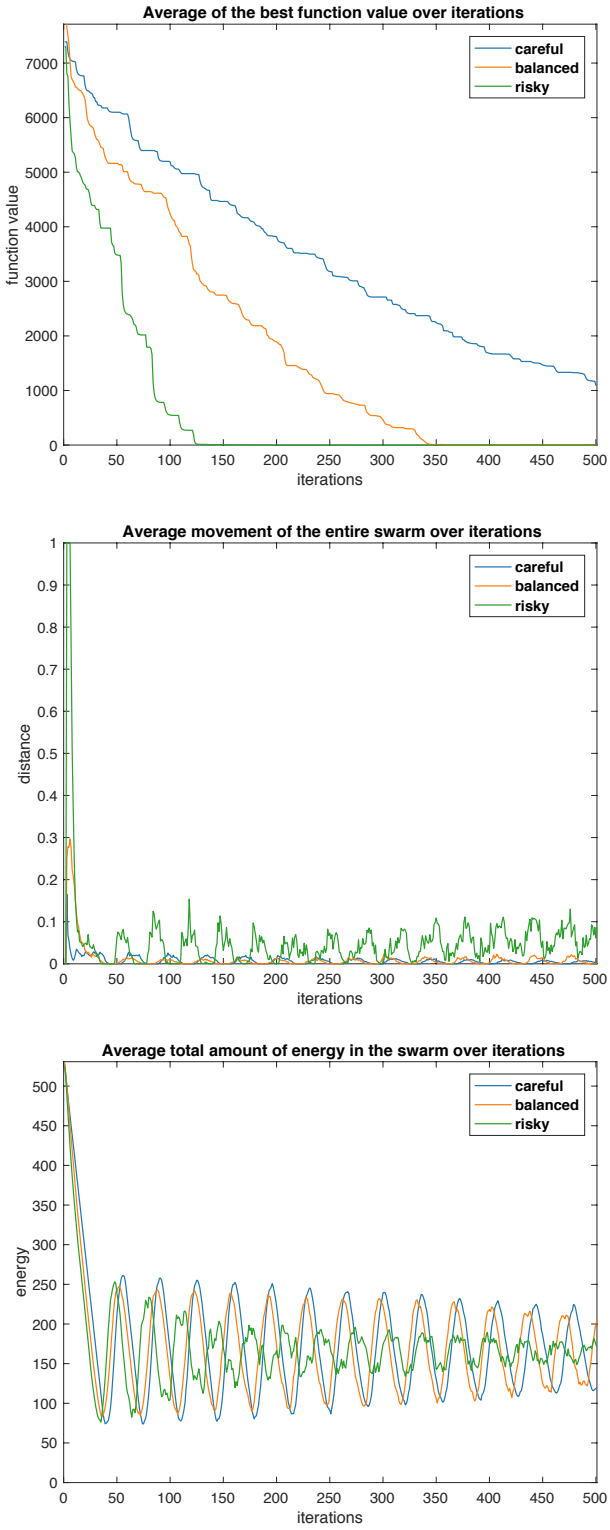


Figure 5.8: Convergence plot, energy plot and movement plot for Rosenbrock function with risk $careful$, $balanced$ and $risky$

5.3.2 Random and Balanced Risk Analysis

In this section, the two types of risk *random* and *balanced* are now compared. *Balanced* rates the two decision factors (cost and profit) equally. Swarm members in the *random* method have all different risk values. We again start with the Sphere function. For this function *balanced* and *random* behave almost identical in the convergence plot and show a rapid convergence. The method *random* reached somewhat faster the optimum. This shows the benefits of different risk levels in the swarm. The method also includes individuals having a risk value in the vicinity of one. These traverse great distances and allow rapid optimization.

More significant is the difference in the moved distance plot. At the beginning, the two methods show a difference in the average movement per iteration. *Balanced* moves in the first iterations more and thus require more energy. Due to the individuals with different risk levels in *random*, the average movement is lower. At the same time, *Random* achieves the optimum faster. This shows that different individuals can be an advantage in a swarm. Individuals with a high risk value fly ahead. Individuals with a low risk value are later attracted by the swarm and take longer. After both methods have reached the optimum, the curves have the same course. The movements are reduced to a minimum. Looking at Table 5.5, one can see that *random* needs a less total distance. Striking is the standard error of the distance for *random*. The error is caused by the different risk value initializations of individuals. Swarms with high average risk values move more than swarms with low risk values (cf. Section 5.3.1).

Considering the energy plot, the uneven course of the *random* method at the beginning stands out. This curve adjusts to the method *balanced*, during the optimization. The various risk values in the swarm prefer different distances. Thus, the energy consumption of each individual is different. On this course, one can see at which time step most of the individual have reached the optimum. From iteration 200 stabilizes the curve. The individuals have reached the optimum, and almost do not move. The movement is at this stage independent of the risk level. The shape of the Sphere function results in negative profit values. This leads the individuals to choose the smaller possible movements. Looking at the total energy consumption in Table 5.5, the difference of the methods is only slightly.

Table 5.5: Results of the Sphere function with risk *balanced* and *random* (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
balanced	0.000	0.000	4430.815	1.157	729.013	2.684
random	0.000	0.000	4425.916	2.541	684.370	8.266

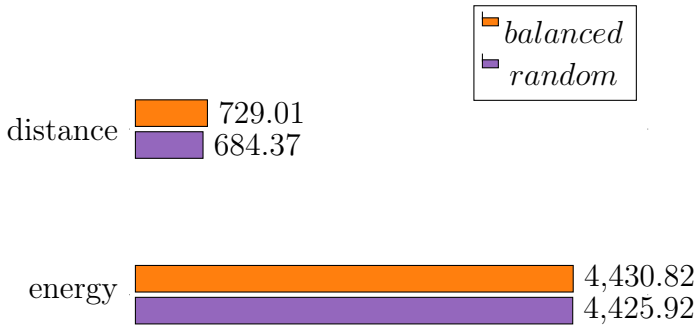


Figure 5.9: Results of the Sphere function with risk *balanced* and *random* (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

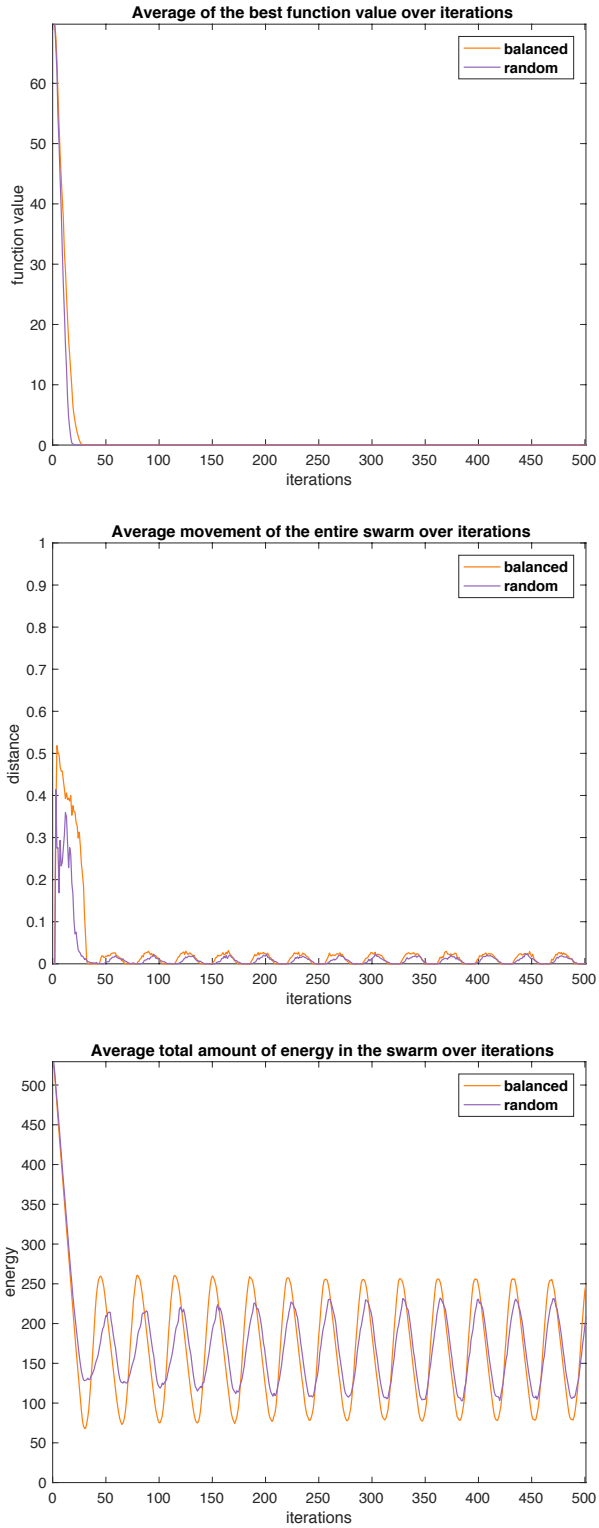


Figure 5.10: Convergence plot, energy plot and movement plot for Sphere function with risk *balanced* and *random*

For the Ackley function, *balanced* and *random* show at the beginning a rapid convergence. *Random* then remains in local optima and needs about 50 iterations to leave this. In this case, *balanced* has the better course. The method starts with a slower convergence than *random*, but the progress is continuously and finally the optimum is reached faster.

The motion plot emphasizes this theory. Both methods start with a similar average distance per iteration. The distance is slightly lower for *balanced*, than e.g. in the Sphere function. This is also due to the local optima. After the first charge cycle, a plateau begins in the convergence plot. Here denotes *random* substantially less motion than *balanced*, although the optimum has not been reached. There are two essential reasons. On the one hand, *random* has a number of individuals who have a very low risk level. These individuals are almost not moving. This leads to the fact that on account of the local optima, these individuals do nothing to improve the function value. In this case, the individuals are useless for the swarm and only consume energy. On the other hand, the different risk values cause a desynchronization of the swarm members. This means that the individuals charge and move no longer at the same time. For functions with local optima, it is better if the swarm fly and charge together. The joint movements can prevent that many individuals gather in local optima, and no one can get away from there. Table 5.6 shows how different the method *random* can be. The average distance is higher for *random*. Considering also the standard error, it is noticeable that this value is influenced by the individual runs. The risk values in the swarm can differ from run to run. Thus, the values are difficult to compare. Nevertheless, it is clear that *random* moves more.

A second disadvantage is the movement, even though the optimum is reached. The behavior occurs in both methods (only delayed). This shows that there are still individuals that try to achieve the optimum. These individuals stay in local optima and consume energy. After a short time this behavior is no longer visible in the moved distance plot. Also in the energy plot, only *balanced* shows a uniform course. For *random*, these movements are easier to see. During the period of the function value plateau, very dissimilar curve can be seen. Here is no unified movement in the swarm. The same can be seen at the end of the optimization. Some individuals try to achieve the optimum, others just hover. Thus, no synchronization prevails in the swarm. In addition, Table 5.6 shows that *random* takes a longer distance and consumes much more energy.

Table 5.6: Results of the Ackley function with risk *balanced* and *random* (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
balanced	0.000	0.554	4531.115	8.800	1008.798	22.184
random	0.001	0.203	4773.944	12.115	1766.415	37.688

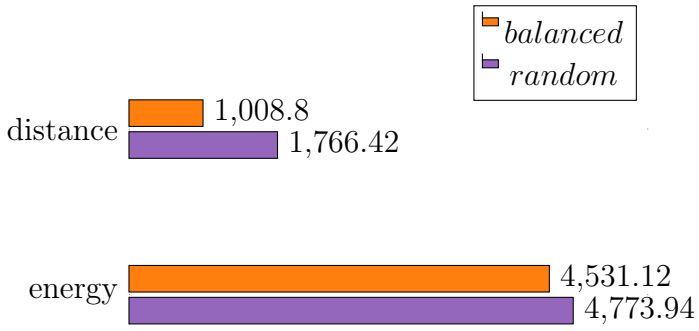


Figure 5.11: Results of the Ackley function with risk *balanced* and *random* (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

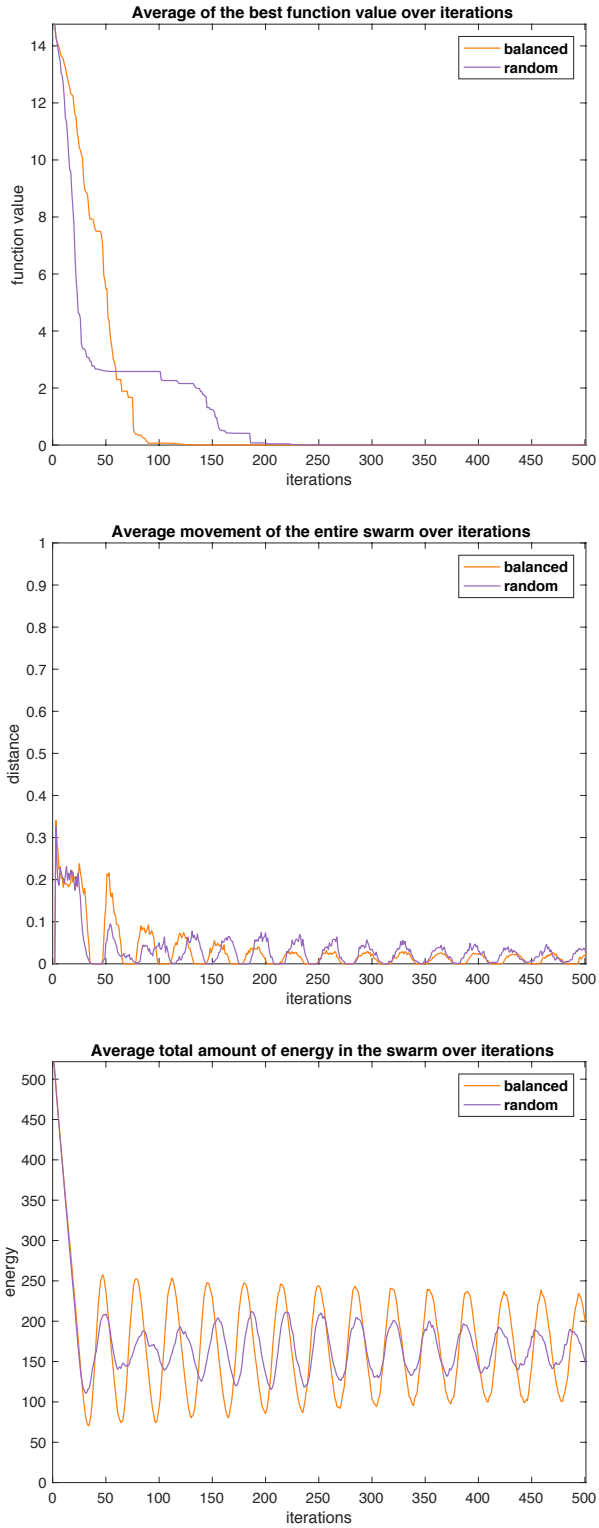


Figure 5.12: Convergence plot, energy plot and movement plot for Ackley function with risk *balanced* and *random*

To conclude this Section, we consider the Rosenbrock function. Here shows *random* and *balanced* a continuous improvement of the function value. Both methods achieve the optimum. The *random* method is about 150 iterations than the *balanced* method. Table 5.7 shows for *balanced* a small deviation from the optimum. Noticeable here is the standard error. A closer look at individual runs shows that some of them end up very far from the optimum. This indicates that small risk values in flat landscapes can be a disadvantage (cf. Section 5.3.1).

Similar to the convergence plot, the average distances are similar. Both methods start with an average distance of 0.3. From the 50th iteration, both methods show only a minimal movement. This behavior can be explained by the landscape of the function. The Rosenbrock function has a very flat landscape. The starting field of the individuals lies partially on a high point. Thus, despite the flat landscape, an initially rapid optimization can be achieved. Only after the first load cycle, at iteration 50, the movement flattens permanently. Here, a small plateau can be seen in both methods (convergence plot). This suggests that the swarms are synchronized and in this period most of the individuals are charging. Caused by the start area, the individuals pull together very quickly. Therefore, according to the charging phase only small movements can be seen. The effect is reinforced by the flat landscape. Although the distance curves are very similar, but the total distances in Table 5.7 differ significantly. This shows that a large part of the swarm acts similar, but still, a lot of unnecessary movements are made by some individuals.

The energy plot shows an interesting progress. In the first 100 iterations, the curves of the methods are almost identical. Only later, the *random* method loses synchronization. This behavior is not visible in the other plots of the function. The amplitude of the *balanced* curve decreases too, but not as strong as for the *random* method. Another effect, in this case, is the displacement of the loading point to the right. This shows that the *random* method has difficulties with flat landscapes. Some individuals (with a low risk value) remain at the optimum, others (with a high risk value) wander around the optimum. This behavior is unnecessary and consumes energy.

Table 5.7: Results of the Rosenbrock function with risk *balanced* and *random* (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
balanced	1.483	68.10	4432.276	7.299	714.238	28.774
random	0.018	0.174	4549.582	12.328	1029.446	41.034

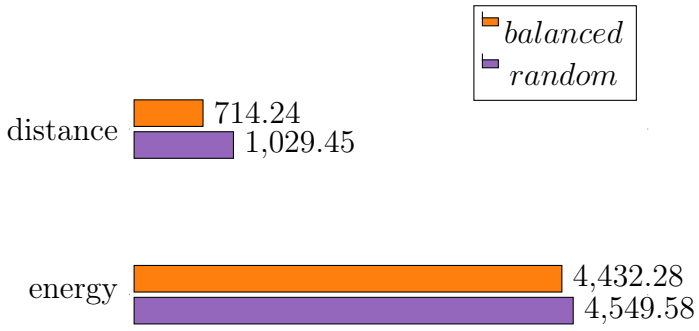


Figure 5.13: Results of the Rosenbrock function with risk *balanced* and *random* (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

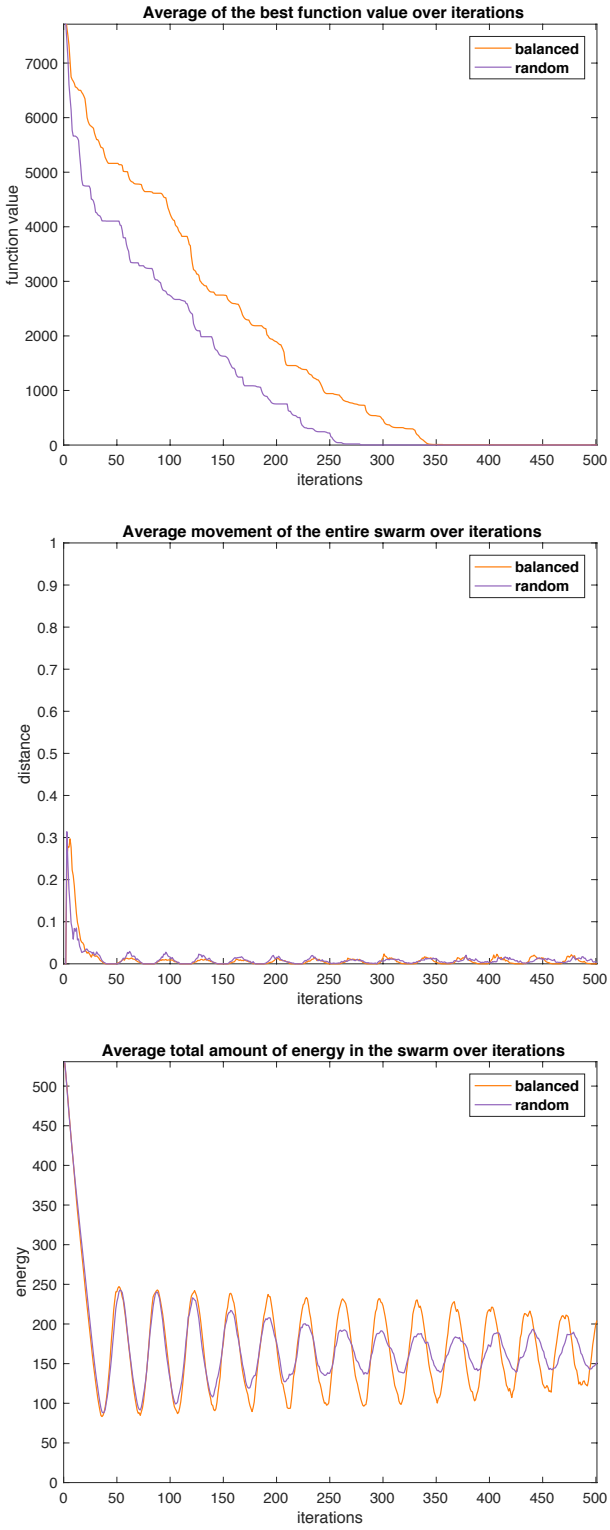


Figure 5.14: Convergence plot, energy plot and movement plot for Rosenbrock function with risk *balanced* and *random*

5.3.3 Random and Adaptive Risk Analysis

In this section, we compare our new *adaptive* EAPSO approach with the *random* method. Both of them include different individuals in the swarm. The difference of *adaptive* is that this method calculates the risk value out of the energy value. This means the risk value change from iteration to iteration. We start again with the Sphere function to validate the concept of our new approach.

In the convergence plot, the *adaptive* method shows first a similar course as *random*. After a short time, the *adaptive* curve is stagnating. From this point, the energy saving behavior of the method is shown. The energy of the individual drops and the distance traveled per iteration decrease. This flattens the progress of the curve. In the further process, "steps" can be seen in the curve. These are caused by the collective moving and loading of the individuals. In contrast, the method *random* shows a completely different movement pattern. Here are always individuals in motion and the curve does not stagnate. Again, the small movements are supported by collecting the swarm. Table 5.8 shows that both methods achieve the optimum with no standard error.

Considering the average distance per iteration, the first advantage of our adaptive EAPSO becomes clear. The *adaptive* method has the ability to choose large distances at the beginning. Thus, all the energy can be used. *Random*, however, only uses an average distance of 0.3. Subsequently, the distance curve flattens quickly for both methods. This occurs even though the optimum is not achieved. The individuals start with an almost full battery. When charging, the batteries are not completely filled. The individuals start when the battery is filled to 50 percent. The resulting risk values again cause shorter movements. In addition, the dense swarm provides only small movements.

The energy plot shows for the *random* method the already explained curve (see Section 5.3.2). The curve of the *adaptive* method, however, shows on the entire time a uniform flow. This confirms the effect of adaptive risk values. Individuals with a low risk value choose short distances, high risk individuals choose greater distances. Caused by this, the individuals move and load almost uniformly.

The plots and the table clearly show that our new method is able to save energy. The disadvantage of the energy saving behavior is the convergence

time. The difference is reflected in the moved distance in Table 5.8. *Adaptive* travels significantly less. The difference in the total amount of energy is, for this simple function, not significant.

Table 5.8: Results of the Sphere function with risk *random* and *adaptive* (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
adaptive	0.000	0.000	4404.301	1.315	568.176	6.851
random	0.000	0.000	4425.916	2.541	684.370	8.266

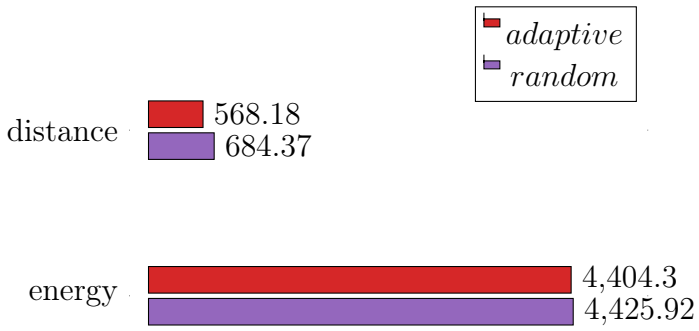


Figure 5.15: Results of the Sphere function with risk *random* and *adaptive* (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

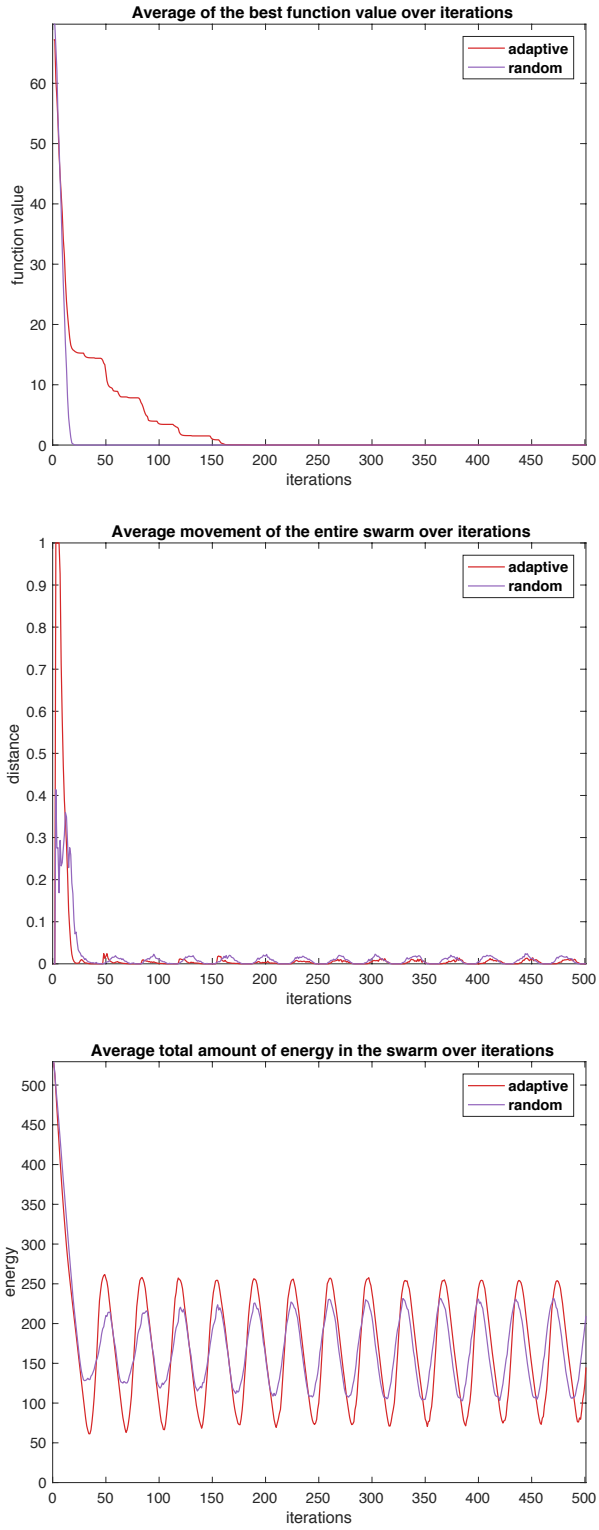


Figure 5.16: Convergence plot, energy plot and movement plot for Sphere function with risk *random* and *adaptive*

The Ackley function shows a similar scene. First, the curves are similar again. After a short time, the *adaptive* method stagnates, and shows to the end the already described "step behavior". Striking here is, that both methods have a plateau at the same function value. At this point, both methods are trapped in a local optimum. *Random* finally reaches twice as fast as *adaptive* the optimum (*random* after 200 iterations and *adaptive* after 400 iterations).

Similar to the Sphere function, *adaptive* shows the required behavior in the average distance plot. In the beginning, large movements can be seen. These are reduced, due to the energy, after the first recharging. The somewhat smaller distance at the beginning is caused by the approximation of the landscape (initially only the actual local optima is known). Here, the difference between the methods is more clearly. Although the *random* method has already reached the optimum, movements still occur. *Adaptive*, however, always shows only very small movements. This decreases even when the optimum is reached. For the Ackley function, Table 5.9 shows the difference of the methods more clearly. The distance traveled by *random* is more than twice as long as for *adaptive*. Also, the difference in energy consumption is obvious.

The next effect is shown in the energy plot. Comparing the two curves in more detail, it becomes apparent that the *adaptive* curve has one charging cycle less. *Random* shows at the beginning and at the end a very jagged curve and low amplitude. Referring the convergence plot, it is noticeable that in the first portion no improvement of the function value is made. This suggests that due to the very different risk levels in the swarm, no uniform motion comes about. *Adaptive* in turn, apart from the decreasing amplitude, shows no change of the curve. Here the trade off between convergence time and energy consumption is significant.

Table 5.9: Results of the Ackley function with risk *random* and *adaptive* (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
adaptive	0.002	0.852	4449.128	5.755	775.828	19.086
random	0.001	0.203	4773.944	12.115	1766.415	37.688

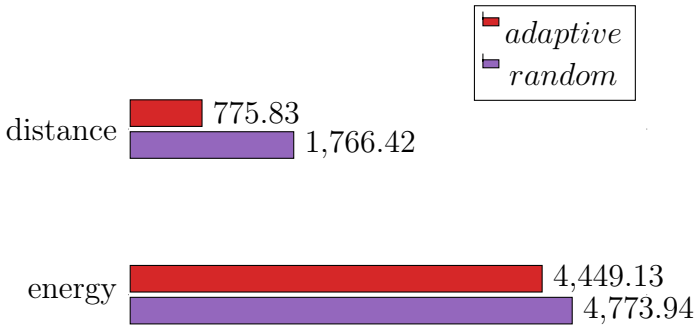


Figure 5.17: Results of the Ackley function with risk *random* and *adaptive* (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

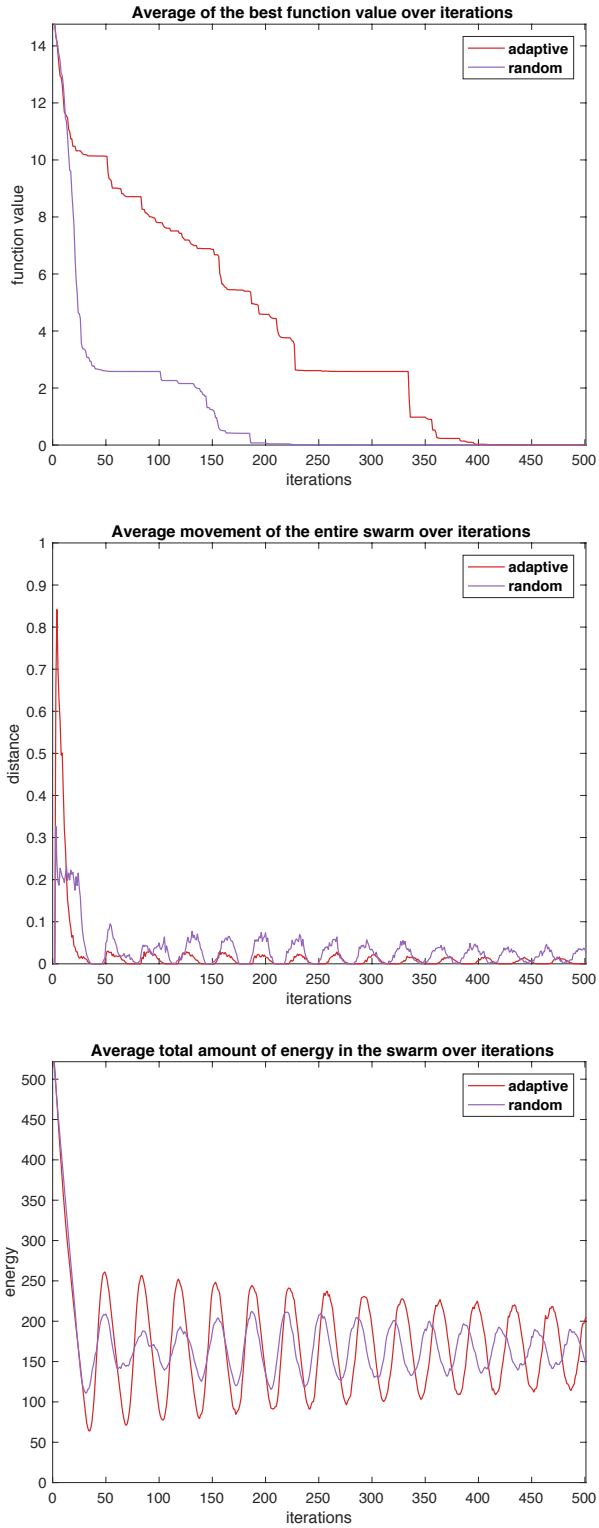


Figure 5.18: Convergence plot, energy plot and movement plot for Ackley function with risk *random* and *adaptive*

Also in this section, we finally consider the Rosenbrock function. For the Rosenbrock function, the method *adaptive* shows a much better result. Here we see that *adaptive*, in the first movement phase, can explore the valley spaciously. As with the other functions, the first charging phase begins after 25 iterations. From here, the "step behavior" can be seen again. The *random* method is slower at the start but reached the optimum almost simultaneously with *adaptive*. In comparison, *adaptive* works here better. Especially with flat landscapes, the use of the entire energy at the beginning is very beneficial. The average obtained fitness value in Table 5.10 shows a better result for the *random* method. The corresponding standard error is smaller for the *adaptive* method. This shows the constant behavior over a number of runs.

The average movement per iteration shows the known behavior of the *adaptive* method. Here all the energy is used. Only after the first charging cycle, the energy saving phase begins. *Random*, on the other hand, chooses only an average distance of 0.3 at the start and thus slows down the convergence. The *random* method shows only a small movement. Looking closely at the *adaptive* curve, one can see almost no more movement. Here a lot of energy is saved. The difference is also reflected in consideration of Table 5.10. The Table shows a difference of over 160 in the total energy consumption of the swarm. Also, the total moved distance is for *random* more than twice as high. Again, the standard error for *adaptive* is in both cases significantly lower.

Also, the energy plot shows a uniform course for the *adaptive* method again. It is striking that even after reaching the optimum this course is not lost. This proves the synchronization of the swarm. After the optimum is reached, the individuals use the energy almost exclusively for hovering. Thus, the individuals load and unload very evenly.

For the flat landscape of the function, we achieve the best result. The consumed energy and the moved distance are both smaller for *adaptive*. At the same time, the convergence time is not inferior.

Table 5.10: Results of the Rosenbrock function with risk *random* and *adaptive* (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
random	0.018	0.174	4549.582	12.328	1029.446	41.034
adaptive	0.067	0.014	4387.826	0.774	433.425	2.552

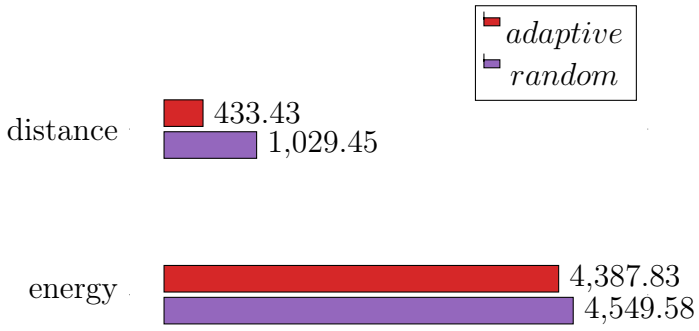


Figure 5.19: Results of the Rosenbrock function with risk *random* and *adaptive* (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

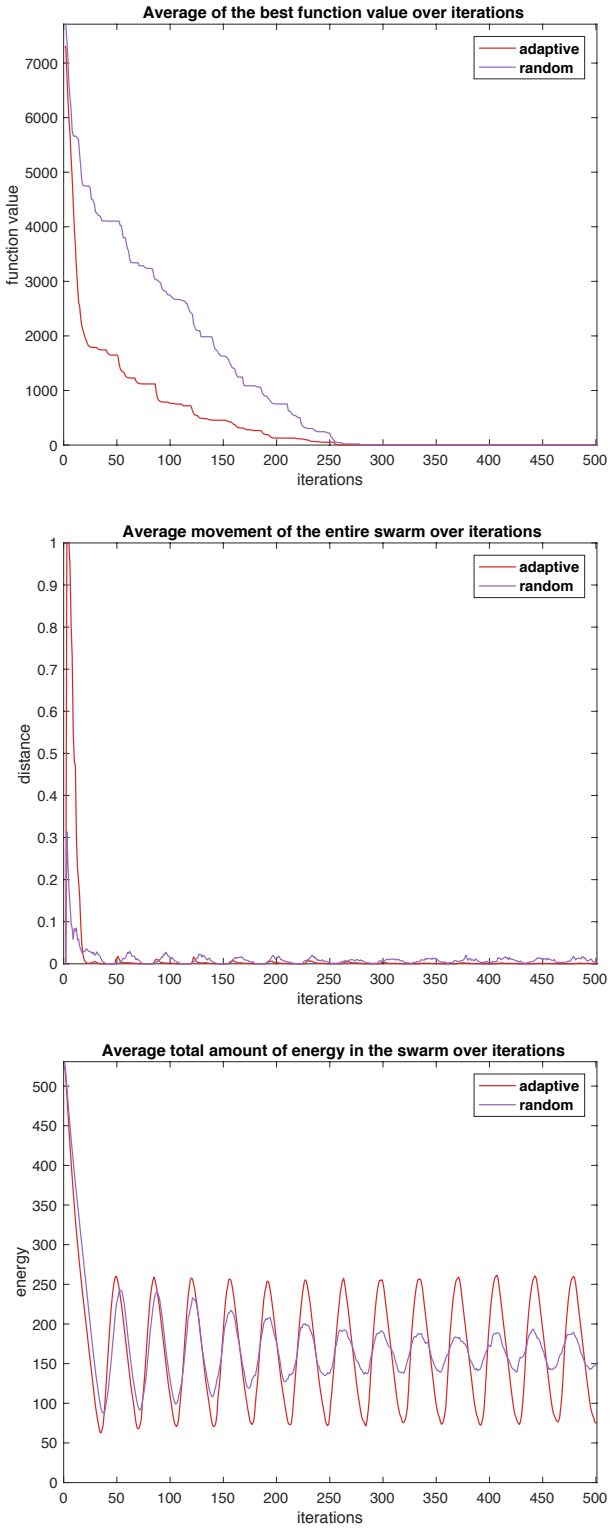


Figure 5.20: Convergence plot, energy plot and movement plot for Rosenbrock function with risk *random* and *adaptive*

5.3.4 Baseline Risk Analysis

Finally, we compare our new *adaptive* EAPSO with the default PSO. We want to show the energy saving features and demonstrate advantages and disadvantages. Therefore we start with the Sphere function.

For the Sphere function, *adaptive* shows the "step behavior", explained in Section 5.3.3. The default PSO method is similar to the *random* method, *balanced* and *risky*. It shows here a clear advantage in the speed of convergence.

Both methods show to the start a similar movement course. In the first cycle of movement, both methods use the maximum distance and travel large distances. We observe that the movement of the individuals in default PSO never stops, even if the PSO has already obtained the optimal solution. In contrast to this, in the *adaptive* EAPSO the individuals reduce their movements (and therefore the energy consumption) to a large extent. Both of the approaches have a cyclic behavior in the distances due to the recharging effect in the model. Since the individuals, all start with a certain high battery level, many of them require a recharging at the same time steps. Considering Table 5.11, the first major difference is clear. The swarm of the default PSO method flies more than four times the distance of *adaptive*. The corresponding standard error is nine times as high as for *adaptive*.

The third plot illustrates the total amount of available energy in the entire swarm. We observe that both of the methods have a cyclic energy level. Starting with a large amount of energy, the individuals get synchronized over the iterations, i.e., they all recharge at the same time steps (at the lower peaks). This effect is more visible in EAPSO than in default PSO. This interesting side effect can be explained by the fact that in the adaptive EAPSO the individuals only move if the trade-off between profit and cost is large enough. The change in the energy curve is clearly visible for the default PSO. Once the optimum has been reached, the amplitude decreases. This continues until the end. This supports the statement that the individuals are always moving. A look at Table 5.11 shows a significantly lower total amount of energy for the *adaptive* method.

Table 5.11: Results of the Sphere function with risk *adaptive* and $defaultPSO$ (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
default	0.000	0.000	4924.679	11.950	2274.928	38.020
adaptive	0.000	0.000	4404.301	1.315	568.176	6.851

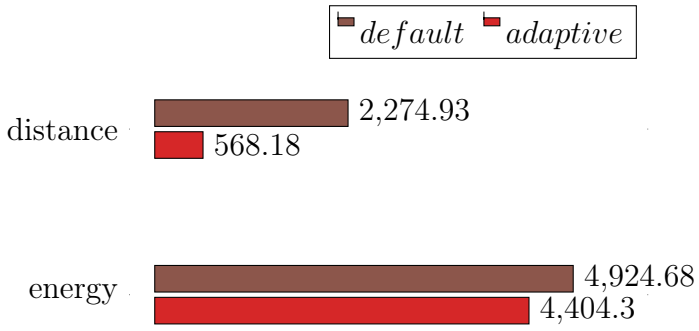


Figure 5.21: Results of the Sphere function with risk *adaptive* and $defaultPSO$ (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

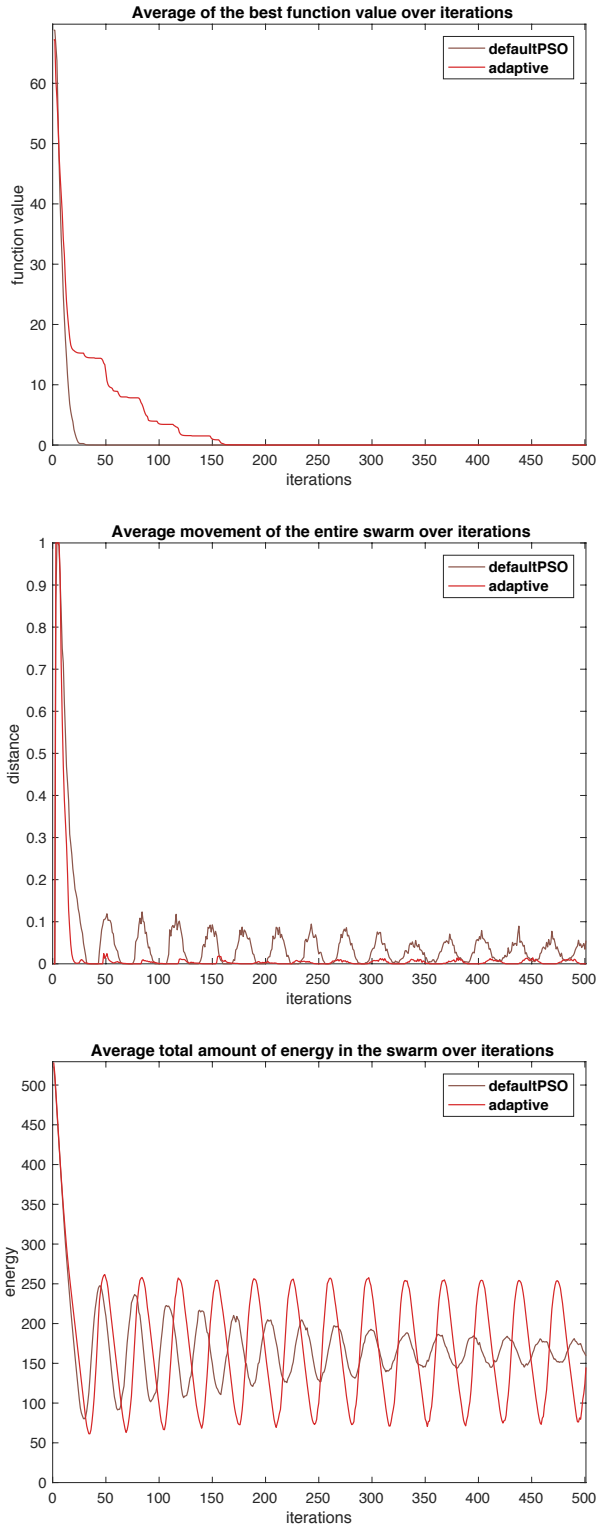


Figure 5.22: Convergence plot, energy plot and movement plot for Sphere function with risk *adaptive* and $defaultPSO$

In the Ackley function, the default PSO has a clear advantage in case of the convergence time. Until the 150th iteration, *adaptive* shows an even faster convergence. After that, the default PSO method shows a steeper curve and reaches ahead of *adaptive* the optimum. In addition, the convergence of *adaptive* is delayed by the pinning in local optima (described in Section 5.3.3). In this case, the standard error of the fitness is higher for the *adaptive* method.

Again, the average movement per iteration is very different. *Adaptive* minimizes the movement after the first recharging as usual. Default PSO shows all the time a relatively high movement. The movements get smaller when the optimum is reached, but there is still movement. During this time a lot of unnecessary energy is consumed. Table 5.12 confirms this statement. The traveled distance is almost three times as long as for the *adaptive* method.

For the Ackely function, the synchronized behavior is less visible than for the other two test problems. The individuals can be trapped in several local optima and build clusters with different properties. By careful observations, we can conclude that the EAPSO individuals require less re-charging cycles than the individuals in default PSO.

Table 5.12: Results of the Ackley function with risk *adaptive* and $defaultPSO$ (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
default	0.001	0.120	4942.563	4.235	2317.692	13.242
adaptive	0.002	0.852	4449.128	5.755	775.828	19.086

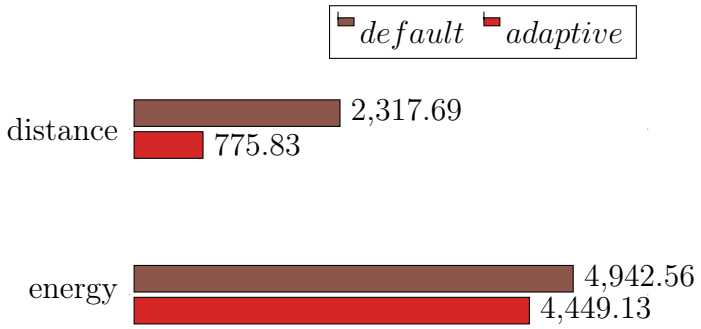


Figure 5.23: Results of the Ackley function with risk *adaptive* and $defaultPSO$ (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

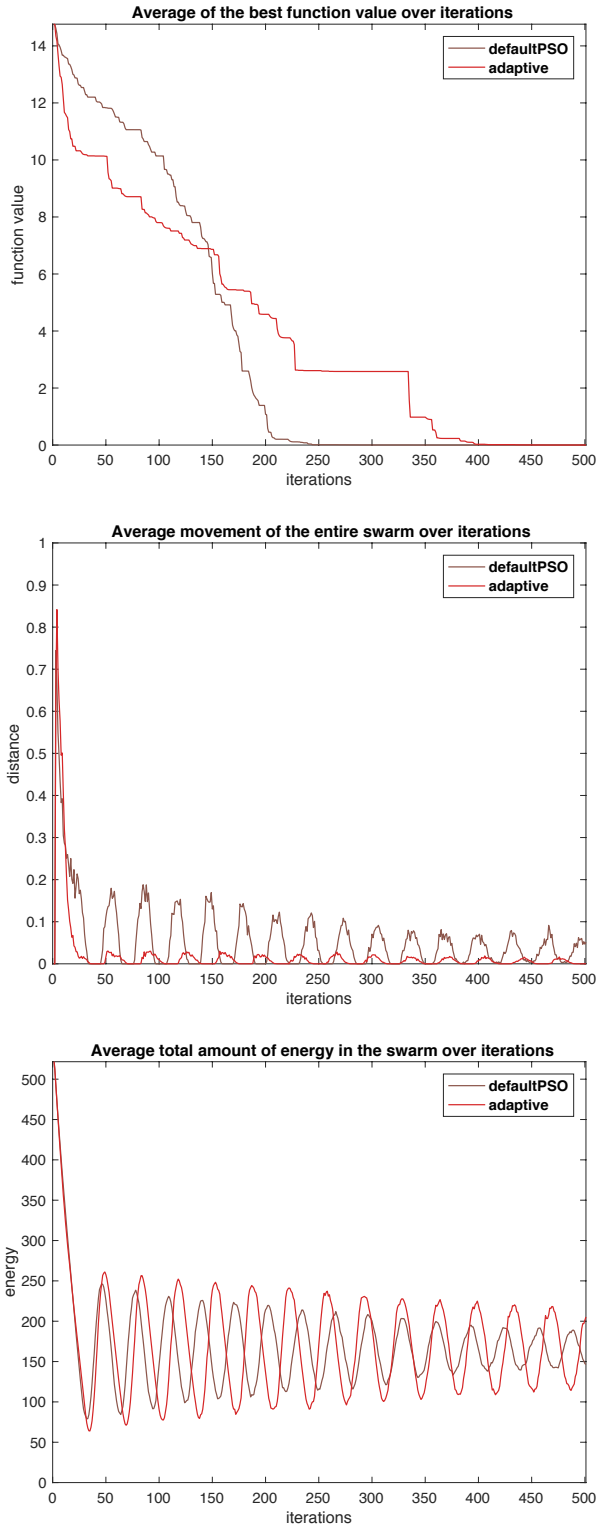


Figure 5.24: Convergence plot, energy plot and movement plot for Ackley function with risk *adaptive* and $defaultPSO$

The last function we consider is Rosenbrock. Both methods show in the first 25th iterations a similar course. From then on, *adaptive* shows again the "step behavior" and the default PSO method reaches the optimum after 50 iterations. The convergence plot also shows the detriment of our adaptive EAPSO concept. Once the first recharging begins, *adaptive* can not keep up with the default PSO.

The advantage, however, shows itself in the average movement plot. Here the range from iteration 200 to the end is interesting. For the two other functions, default PSO shows a continuous movement. However, it is always a recharge phase recognizable. For the Rosenbrock function, a different scene emerges. No unified movement can be seen. The difference of the total moved distance is here the greatest. According to Table 5.13, default PSO needs more than five times the distance of the *adaptive* method.

The observation is supported by energy plot. After the 250th iteration, no oscillation can be seen. During this time no uniform motion longer takes place. Some individuals stay at the optimum and only consume the energy for the hovering. Other individuals move around the optimum and consume unnecessary power. Here, the biggest advantage of our algorithm becomes clear. With the calculation of the risk value from the energy, the movement is reduced to a minimum. Again, Table 5.13 supports this observation. The total energy consumption for *adaptive* is significantly lower.

Table 5.13: Results of the Rosenbrock function with risk *adaptive* and $defaultPSO$ (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
default	0.000	0.000	4929.129	9.580	2291.180	31.136
adaptive	0.067	0.014	4387.826	0.774	433.425	2.552

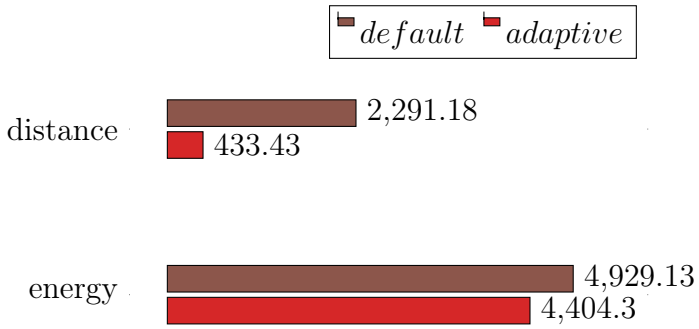


Figure 5.25: Results of the Rosenbrock function with risk *adaptive* and $defaultPSO$ (median values). "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

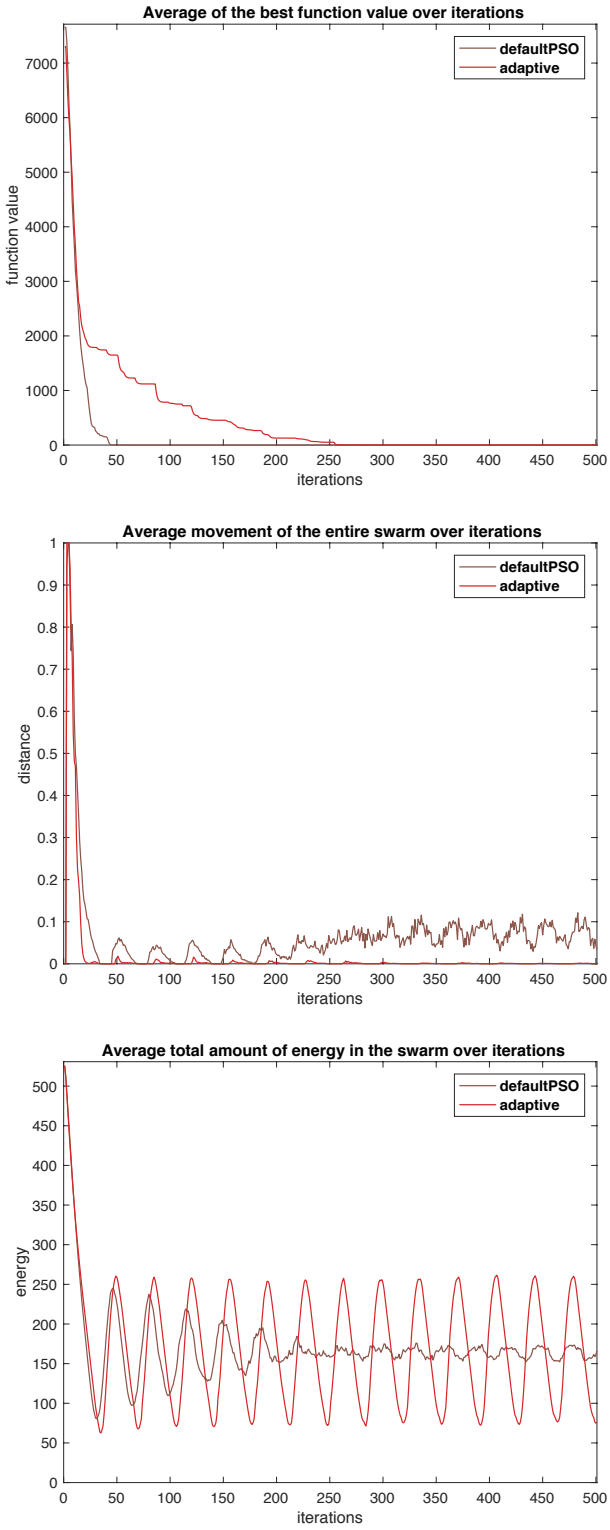


Figure 5.26: Convergence plot, energy plot and movement plot for Rosenbrock function with risk *adaptive* and $defaultPSO$

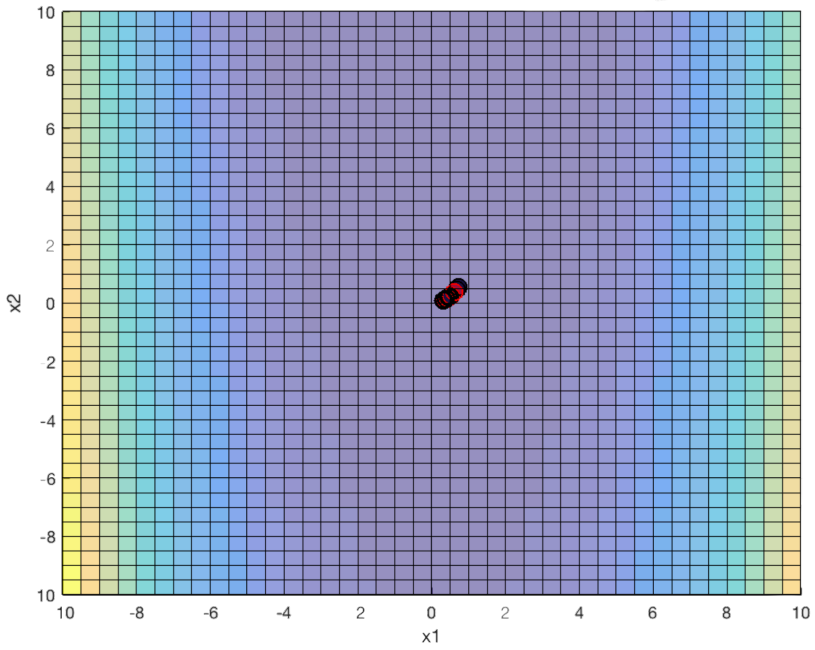
5.3.5 Summary

In this section, we summarize the results and have a look at some detailed results. Table 5.14 shows the results after 500 iterations. All the experiments are run for 30 times and the median values, and the corresponding standard errors are reported. In all the experiments the *careful* method delivers the worst fitness value as expected. In this case, the individuals always select the closest better individual as their global best and therefore they save lots of energy and distance. Considering the Sphere function, we observe that all the other EAPSO variants can find the optimal solution where the particles in default PSO move the largest distance and consume more energy than the others. Among the EAPSO approaches, the *adaptive* method saves the most amount of energy, while *risky* has the highest energy consumption. Considering the Ackley function with lots of local optima, the default PSO obtains the best result in term of the fitness value (and the corresponding standard error values). The *adaptive* variant is not as good as the other variants, nevertheless, its distance and the energy values are the best among the others. The same results can be observed in Rosenbrock function with a large flat plateau. Due to the small amount of profit which can be obtained in a local neighborhood, the individuals reduce the amount of movement and save energy while not making the effort of moving. This leads to a degradation in the function value.

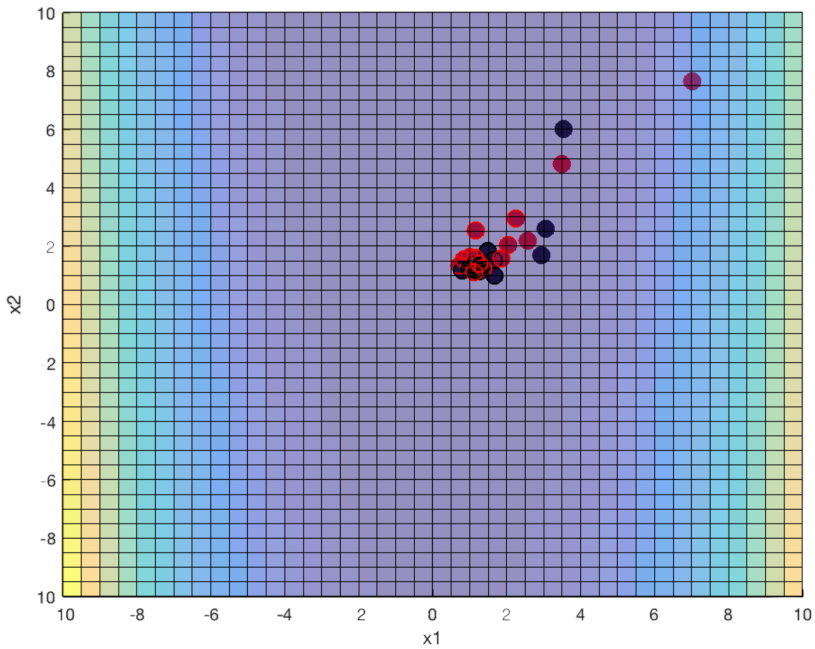
The analysis revealed, especially in Section 5.3.4, that the adaptive EAPSO method avoids unnecessary movements. To illustrate this, we show in Figure 5.27 the two methods adaptive EAPSO and default PSO in the Rosenbrock function after 450 iterations. Both methods have reached the optimum. The individuals in the adaptive EAPSO method are crowded. In the default PSO method, the individuals are widely distributed. This shows that the individuals of the default PSO method move on, even though the optimum is reached.

Table 5.14: Results for the three test problems with 30 individuals and 500 iterations (median values and standard errors (std)). "fitness" refers to the best function value obtained by the swarm, "energy" and "distance" indicate the total amount of energy and the distance moved by the swarm

risk r_i	fitness	± std	energy	± std	distance	± std
			Sphere			
default	0.000	0.000	4924.679	11.950	2274.928	38.020
careful	5.598	0.485	4375.483	1.499	415.959	5.337
balanced	0.000	0.000	4430.815	1.157	729.013	2.684
risky	0.000	0.000	4449.644	1.203	799.311	3.406
random	0.000	0.000	4425.916	2.541	684.370	8.266
adaptive	0.000	0.000	4404.301	1.315	568.176	6.851
			Ackley			
default	0.001	0.120	4942.563	4.235	2317.692	13.242
careful	11.61	0.162	4505.048	3.383	935.639	9.089
balanced	0.000	0.554	4531.115	8.800	1008.798	22.184
risky	0.001	0.171	4993.991	6.649	2491.168	25.109
random	0.001	0.203	4773.944	12.115	1766.415	37.688
adaptive	0.002	0.852	4449.128	5.755	775.828	19.086
			Rosenbrock			
default	0.000	0.000	4929.129	9.580	2291.180	31.136
careful	1095	76.57	4391.300	2.108	491.280	8.549
balanced	1.483	68.10	4432.276	7.299	714.238	28.774
risky	0.102	0.296	4899.941	17.155	2179.118	58.146
random	0.018	0.174	4549.582	12.328	1029.446	41.034
adaptive	0.067	0.014	4387.826	0.774	433.425	2.552



(a) adaptive EAPSO



(b) default PSO

Figure 5.27: Rosenbrock function after 450 iterations with risk *adaptive EAPSO* and *default PSO*

Additional Observation

Finally, we highlight a result that is not apparent from the plots. By considering single runs, we observed some individuals who are separated from the main swarm. These individuals have subsequently formed their own swarm or cluster.

On investigation, we noticed that the individuals in a cluster have similar risk levels. Due to the similar risk values, these individuals also selects similar neighborhood sizes. Thus, it happens that seceded individuals explore a part of the landscape, independently of the other individuals. Due to the flat landscape of the Rosenbrock function, there are no disturbing local optima that may block this behavior. For this behavior, a swarm with different risk types (e.g. *random* or *adaptive*) is required. Only with different characters, it is possible to produce this behavior. Especially for the type *adaptive* it happens that individuals move between clusters or clusters merge together. This behavior can be observed when a group of individuals starts after recharging. Due to the energy level, the risk is high, and the individuals move towards clusters with similar risk levels.

6 Conclusion and Future Work

This chapter summarizes the thesis and outlines possible improvements for future work.

6.1 Conclusion

This thesis introduced the EAPSO (Energy Aware PSO) method as a search mechanism for a swarm of aerial micro-robots. For that, we have defined some objectives. Based on these objectives, we conclude this thesis.

> **Objective 1**: Modeling a general movement behavior for the individuals in a swarm

We build a movement model for the individuals in a swarm with different states and actions. Each individual has the ability to choose between different states and actions. We have defined two states: "Ground" and "Air". Each state has its own actions. In state "Ground", the following actions can be selected: start and charge. The state "Air" completes the model with the following actions: fly, hover, land, and wait. With the defined states and actions, the necessary range of functions were described.

> **Objective 2**: Building a simplified model of energy consumption for the individuals, based on FINken-III

We have introduced a simplified model of energy consumption for the individuals, based on the aerial micro-robot FINken-III. The energy of an individual

is defined in units per iteration and several constants represent the energy consumption of different actions.

Objective 3: PSO-based search mechanism which considers the amount of energy consumption for each individual

The proposed model is built upon the default PSO with an additional multi-criteria decision making aspect for the individuals which make a decision before starting a movement. The decision is made based on two objectives, profit in terms of the overall gain in the search process and cost in terms of the energy consumption. The profit is calculated with an approximation of the landscape. We have used weighted sum approach and an adaptive version for the decision making.

The algorithm uses the simplified model of energy consumption, a "k-Nearest-Neighbor" neighborhood topology and the discrete time flight (from PSO) in the search mechanism.

Objective 4: Implementation of the Energy Aware PSO approach for evaluation and future projects

The implementation includes all described components of the EAPSO approach and the structure is based on the standard PSO. The main components are the swarm and the particle class. The movement model is implemented in the state class and the energy class describes the model of energy consumption for the individuals. The multi-criteria decision making process is realized by neighborhood and the leader selection class.

Objective 5: Evaluation of different EAPSO methods, to show whether the methods work out as desired

In the experiments, the different methods (*careful*, *balanced*, *risky*, *random* and *adaptive*) are examined. We have shown how the methods work in different landscapes and pointed out some advantages and disadvantages. For that, the convergence of the function value, the moved distance per iteration

and the energy in the system are measured. The experiments on three test problems (Sphere, Ackley and Rosenbrock) show that EAPSO can be used as a search mechanism for swarm of aerial micro-robots and integrating the decision making process in the optimization can extensively reduce the energy consumption, while the quality of search will be influenced.

6.2 Future Work

This work has opened a large number of research questions for future work. Below we present some ideas that can help to improve the algorithm.

As an extension of the existing multi-criteria decision making a learning system can be used. While the copters are flying, they can collect data with various sensors for later evaluation. One option is to learn the PSO parameters w, C_1 and C_2. These parameters are constant for the purpose of simplification in our experiments. With the help of the previous movements, sensor data, and some decision rules, it is possible to learn the parameters of the PSO formula. Thus, an adaptation to the landscape is possible. Another variation is the risk value itself, to incorporate as a kind of learning factor. A problem that becomes clear here is the amount of sensor data that accumulates the copter itself or from his neighborhood. To process and filter the important data presents a challenge.

Another idea is to monitor permanently the sensor values (especially the height sensor) during the flight. A disadvantage of the current model is that only the start and the end point of movement is known. The structure of the landscape, which is overflown in the movement is not stored or analyzed. The idea is to analyze the covered distance after the actual movement and verify whether a better function value is found on the route. If this is the case, the copter can decide to plan the next flight in the direction of this point. Alternatively, the copter can compare the actual target point with the new one and decide to terminate the flight. Another advantage, resulting from the analysis of the trajectory is the better understanding of the landscape. The more known points of the landscape are available, the better is the approximation of an unknown point. As described in 3.7, the approximation of the environment is an important part of decision making. Therefore, in future work, this part should be further explored.

To improve the approximation, an adaptive approximation size can be used. Thus, each individual can decide if he increases or decreased the number of points for the approximation. Particularly useful is this mechanism at the Ackley function. Individuals can detect that the error is too large, and increase the amount of points. In this case, not only the local optimum is approximated, but also other parts of the function. In the Sphere function, individuals can use more points at the start and less later. Thus, the error at the beginning is minimized and computing capacity can be saved.

Another approach is to work with multiple tasks. Here it is possible to specify multiple targets (waypoints) for the entire swarm that needs to be reached. These waypoints can be local optima in a function or an individual function. In this case, the entire swarm flies from target to target. Another possibility is to give the swarm or each individual the ability to decide independently what goal they want to track. Again, the risk value can influence this decision. Individuals with low energy or small risk values can work on nearby targets. In relation to the *adaptive* approach, the goal of an individual may change over time. Individual with a high level of risk can decide to choose a distant goal. After some time, when the power runs low and the individual is in the vicinity of another target, it can decide to visit a new destination. In this approach, we need a threshold providing information about the processing status of a task. If a task is almost completely fulfilled (e.g. the minimum is already found), it is unattractive for other individuals and they look for another destination.

The following idea deals with the model of the movement and the possible actions, described in Section 3.1. So far, the copter performs every move, even if they are very small. Here it is possible to let the copter decide whether the landing or the hovering action is more energy efficient than the fly. For this purpose, the movement cost and the profit must be considered. In the event that the cost of a small fly is too high in relation to the profit, the copter can decide not to fly. Thus, the copter can save some energy. However, there are two problems. On the one hand, it may happen that the individuals can only decide between small movements and no longer fly to the optimum. On the other hand, the decision whether a flight is too expensive or the profit is too small, is not easy.

Another approach is to integrate the risk value or the energy into the PSO formula. One possible option is the constriction factor proposed by [Clerc and Kennedy, 2002] and [Eberhart and Shi, 2000]. This is actually used

to improve the control of the swarm. For that, the equation from Section 2.1 is multiplied by the constriction factor $\mathcal{X}$. The factor is usually a constant value, but it is possible to use the energy level as a factor. Thus, the movement can be slowed down and the individuals move dependent on their energy level.

$$\vec{v}_i(t+1) = \mathcal{X}[\omega\vec{v}_i(t) + C_1\phi_1(\vec{P}_{best} - \vec{x}_i(t)) + C_2\phi_2(\vec{x}_g - \vec{x}_i(t))] \tag{6.1}$$

The evaluation has shown that the charge level has a large effect. A disadvantage which has been shown was the loading of the individuals. Caused by the fact that the individuals recharge up to a power level of 50 percent, the individuals are not able to fly long distances. Here it is useful to raise the percentage. Another possibility is to let the individuals decide whether a longer recharging time is useful or not. Factors can be the distance to the (calculated) optimum, the distance to the other individuals or movements in the past. Thus, the individuals can reach the target quickly. Once this is achieved, the energy-saving characteristic can be used.

These and other approaches can help to make the presented algorithm better and more efficient. The results also motivate further research in this area. The next step is to work on the assumptions and implement the search mechanism on our Hardware platform (FINken-III).

Bibliography

[Amory et al., 2013] Amory, A., Meyer, B., Osterloh, C., Tosik, T., and Maehle, E. (2013). Towards fault-tolerant and energy-efficient swarms of underwater robots. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1550–1553. IEEE.

[Amory et al., 2014] Amory, A., Tosik, T., and Maehle, E. (2014). A load balancing behavior for underwater robot swarms to increase mission time and fault tolerance. In *Parallel Distributed Processing Symposium Workshops (IPDPSW)*, pages 1306–1313.

[Bermes, 2010] Bermes, C. (2010). *Design and dynamic modeling of autonomous coaxial micro helicopters*. PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 18847, 2010.

[Bouabdallah, 2007] Bouabdallah, S. (2007). *Design and control of quadrotors with application to autonomous flying*. PhD thesis, Ecole Polytechnique Federale de Lausanne.

[Clerc and Kennedy, 2002] Clerc, M. and Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73.

[Eberhart and Shi, 2000] Eberhart, R. C. and Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 84–88. IEEE.

[Engelbrecht, 2005] Engelbrecht, A. P. (2005). *Fundamentals of Computational Swarm Intelligence*. Wiley.

[Gazi and Passino, 2011] Gazi, V. and Passino, K. M. (2011). *Swarm stability and optimization*. Springer Science & Business Media.

[Grodzevich and Romanko, 2006] Grodzevich, O. and Romanko, O. (2006). Normalization and other topics in multi-objective optimization.

[Hattenberger et al., 2014] Hattenberger, G., Bronz, M., and Gorraz, M. (2014). Using the paparazzi uav system for scientific research. In *IMAV 2014, International Micro Air Vehicle Conference and Competition 2014*, pages pp–247.

[Hayes, 2002] Hayes, A. (2002). How many robots? group size and efficiency in collective search tasks. In *6th Int. Symp. on Distributed Autonomous Robotic Systems, DARS 2002*, pages 289–298.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization, ieee international of first conference on neural networks.

[Kennedy and Eberhart, 2001] Kennedy, J. and Eberhart, R. C. (2001). *Swarm Intelligence*. Morgan Kaufmann.

[Kiranyaz et al., 2014] Kiranyaz, S., Ince, T., and Gabbouj, M. (2014). *Multidimensional particle swarm optimization for machine learning and pattern recognition*. Springer.

[Kumar and Michael, 2012] Kumar, V. and Michael, N. (2012). Opportunities and challenges with autonomous micro aerial vehicles. *The International Journal of Robotics Research*, 31(11):1279–1291.

[Kushleyev et al., 2013] Kushleyev, A., Mellinger, D., Powers, C., and Kumar, V. (2013). Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300.

[Marler and Arora, 2010] Marler, R. T. and Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41(6):853–862.

[Mei et al., 2005] Mei, Y., Lu, Y.-H., Hu, Y., and Lee, C. (2005). A case study of mobile robot's energy consumption and conservation techniques. In *12th IEEE International Conference on Advanced Robotics, ICAR 2005*, pages 492–497.

[Mei et al., 2006] Mei, Y., Lu, Y.-H., Hu, Y., and Lee, C. (2006). Deployment of mobile robots with energy and timing constraints. *Robotics, IEEE Transactions on*, 22(3):507–522.

[Mettler, 2001] Mettler, B. F. (2001). Modeling small-scale unmanned rotorcraft for advanced flight control design.

[Michael et al., 2012] Michael, N., Shen, S., Mohta, K., Mulgaonkar, Y., Kumar, V., Nagatani, K., Okada, Y., Kiribayashi, S., Otake, K., Yoshida, K., et al. (2012). Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841.

[Miettinen, 1999] Miettinen, K. (1999). *Non-linear Multiobjective Optimization*. Kluwer, International Series in Operations Research and Management Science.

[Moscibroda et al., 2006] Moscibroda, T., von Rickenbach, P., and Wattenhofer, R. (2006). Analyzing the energy-latency trade-off during the deployment of sensor networks. In *IEEE International Conference on Computer Communications*, pages 1–13.

[Moutinho and Hutcheson, 2011] Moutinho, L. and Hutcheson, G. D. (2011). *The SAGE dictionary of quantitative management research*. Sage.

[Mulgaonkar and Kumar, 2014] Mulgaonkar, Y. and Kumar, V. (2014). Autonomous charging to enable long-endurance missions for small aerial robots. In *SPIE Defense+ Security*, pages 90831S–90831S. International Society for Optics and Photonics.

[O'Hara et al., 2006] O'Hara, K., Nathuji, R., Raj, H., Schwan, K., and Balch, T. (2006). Autopower: toward energy-aware software systems for distributed mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2757–2762.

[Pines and Bohorquez, 2006] Pines, D. J. and Bohorquez, F. (2006). Challenges facing future micro-air-vehicle development. *Journal of aircraft*, 43(2):290–305.

[Purshouse et al., 2014] Purshouse, R. C., Deb, K., Mansor, M. M., Mostaghim, S., and Wang, R. (2014). A review of hybrid evolutionary multiple criteria decision making methods. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1147–1154.

[Roberts et al., 2008] Roberts, J., Zufferey, J.-C., and Floreano, D. (2008). Energy management for indoor hovering robots. In *Intelligent Robots and Systems (IROS)*, pages 1242–1247.

[Steup et al., 2016] Steup, C., Mai, S., and Mostaghim, S. (2016). Evaluation platform for micro aerial indoor swarm robotics. Technical Report 3, University of Magdeburg, Faculty of Computer Science. to be published.

[Stirling and Floreano, 2013] Stirling, T. and Floreano, D. (2013). Energy-time efficiency in aerial swarm deployment. In Martinoli, A. e. a., editor, *Distributed Autonomous Robotic Systems*, volume 83 of *Springer Tracts in Advanced Robotics*, pages 5–18. Springer Berlin Heidelberg.

[Stirling et al., 2010] Stirling, T., Wischmann, S., and Floreano, D. (2010). Energy-efficient indoor search by swarms of simulated flying robots without global information. *Swarm Intelligence*, 4(2):117–143.

[Trianni, 2008] Trianni, V. (2008). *Evolutionary Swarm Robotics: Evolving Self-Organising Behaviours in Groups of Autonomous Robots (Studies in Computational Intelligence) (Studies in Computational Intelligence)*. Springer Publishing Company, Incorporated, 1 edition.

[Valenti et al., 2007] Valenti, M., Bethke, B., How, J., de Farias, D., and Vian, J. (2007). Embedding health management into mission tasking for uav teams. In *American Control Conference*, pages 5777–5783.

[Valentini et al., 2015] Valentini, G., Hamann, H., and Dorigo, M. (2015). Efficient decision-making in a self-organizing robot swarm: On the speed versus accuracy trade-off. In et al., R. B., editor, *Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1305–1314.

[Wahby et al., 2015] Wahby, M., Weinhold, A., and Hamann, H. (2015). Revisiting BEECLUST: Aggregation of swarm robots with adaptiveness to different light settings. In *Int. Conf. on Bio-inspired Information and Communications Technologies*.