D6.3 – INITIAL CPS SYSTEM DESIGN OPTIMIZATION AND FITNESS FUNCTION DESIGN GUIDELINES

Deliverable ID	**D6.3**
Deliverable Title	**Initial CPS System Design Optimization and Fitness Function Design Guidelines**
Work Package	**WP6**
Dissemination Level	**PUBLIC**
Version	**1.0**
Date	**30-06-2018**
Status	**Final**
Lead Editor	**Midhat Jdeed (UNIKLU)**
Main Contributors	**Arthur Pitman (UNIKLU)**

Published by the CPSwarm Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731946.

Document history

Version	Date	Author(s)	Description
0.1	2018-04-04	Midhat Jdeed (KLU)	Initial TOC
0.2	2018-04-06	Midhat Jdeed (KLU)	Assignment of partners to sections
0.3	2018-05-09	Midhat Jdeed (KLU)	CPS design section
0.4	2018-05-09	Midhat Jdeed (KLU)	Evolutionary optimization section
0.5	2018-05-18	Midhat Jdeed (KLU)	Fitness function design guidelines section
0.6	2018-06-04	Midhat Jdeed (KLU)	Integration of partner contributions
0.61	2018-06-07	Wilfried Elmenreich (KLU)	Updating the fitness function design guidelines section
0.62	2018-06-12	Arthur Pitman (KLU)	Further integration of partner contributions
1.0	2018-06-30	Arthur Pitman (KLU)	Final version

Internal Review History

Review Date	Reviewer	Summary of Comments
2018-06-20	Regina Krisztina Bíró (SLAB)	Few modifications and comments.
2018-06-27	Etienne Brosse (SOFTEAM)	Minor comments.

Table of contents

Contents

1 Executive summary

Deliverable D6.3 - *Initial CPS System Design Optimization and Fitness Function Design Guidelines* describes the optimization of CPSs using heuristic search approaches and methods for assessing their performance. Special emphasis is given to the development of fitness functions that guide the optimization process. First, this deliverable presents a review of typical optimization methods and evolutionary optimization. Following this, it outlines existing approaches for designing fitness functions and highlights the main challenges. It then considers how this process can be streamlined using the CPSwarm Workbench. Finally, a case study on the Search and Rescue (SAR) scenario is introduced to examine the effectiveness of the technique.

2 Introduction

2.1 Scope

This deliverable considers the design of optimal swarms of Cyber Physical Systems (CPSs) and the corresponding fitness function used in the optimization process and how this can be accomplished using the CPSwarm Workbench. This deliverable does not cover the implementation of the interface between the simulation and optimization environments, such as FREVO and external simulators, in the CPSwarm Workbench. This interface will be covered in deliverable D6.5 'Initial integration of external simulators'.

2.2 Document organization

The rest of this deliverable is structured as follows. First, it presents a review of optimization techniques for swarms of Cyber-Physical Systems focusing on evolutionary approaches. Then, FREVO - an evolutionary optimization tool - is introduced together with the main steps for optimizing the design of swarms of Cyber-Physical Systems. Following this, the deliverable presents an outline of the fitness functions which guide the optimization process together with the steps to design them in the CPSwarm Workbench. Finally, a case study on the Search and Rescue scenario is presented which addresses the effect of fitness function design on the performance of swarms of Cyber-Physical Systems.

2.3 Related documents

ID	Title	Reference	Version	Date
[D3.1]	Initial System Architecture & Design Specification	D3.1	1.0	M8
[D6.1]	Initial Simulation Environment	D6.1	1.0	M9
[D5.2]	Initial CPSwarm Modelling Tool	D5.2	1.0	M9
[D5.1]	CPSwarm Modelling Language Specification	D5.1	1.0	M12
D3.2	Reference to this deliverable is made in this document			
D2.1	Reference to this deliverable is made in this document			
D2.2	Reference to this deliverable is made in this document			
D4.5	Reference to this deliverable is made in this document			
D6.5	Reference to this deliverable is made in this document			

3 CPS design optimization (KLU)

3.1 Designing swarms of Cyber-Physical Systems (CPSs)

The behaviour and the interactions among CPS swarm members remain a complex topic especially in environments where dynamic interactions occur. In particular, it is often difficult to derive the requirements for an individual member within a swarm from the desired global behaviour [1]. Individual theories allow formal descriptions of different aspects of CPS design. Those theories comprise physical, technical, and organizational perspectives at different levels of detail. As shown with a conceptual map in Figure 1, considering even only a subset of topics surrounding the design of CPSs leads to a tremendous increase in complexity. Although this list is far from complete, the need for design and deployment methodologies for CPSs is clearly visible [2].

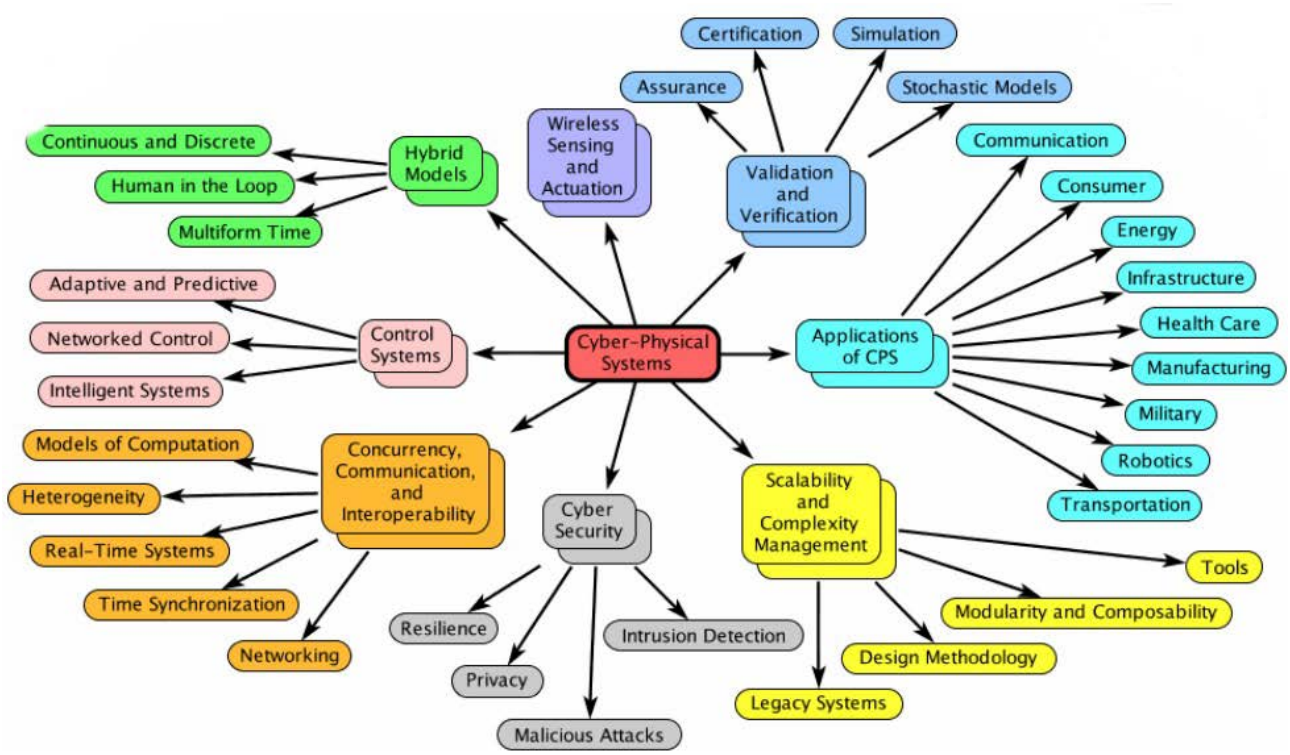

Figure 1: The complexity of designing CPSs

As outlined in [3], the CPSwarm project provides a toolset for the CPS design process (see Figure 2). Initially, a Modelling Library - consisting of formal representations of CPS subsystems, CPS base functions, security recipes, behaviour routines, swarm and self-organization algorithms, and human-to-CPS interaction patterns - is constructed as a cornerstone upon which the project is founded. Designing a CPS with this approach assumes that the initial problem definition is known and properly defined during the modelling phase. Then, once the conditions for a CPS are set, the system emerges over iterations though optimization based on simulation. In each iteration, the simulation results are validated against the acceptance criteria and restricted by a defined fitness function. Once the design and the engineering of the algorithms for the CPSs are complete, the final stage is to start the deployment by automatically transforming the generated code to hardware specific requirements.

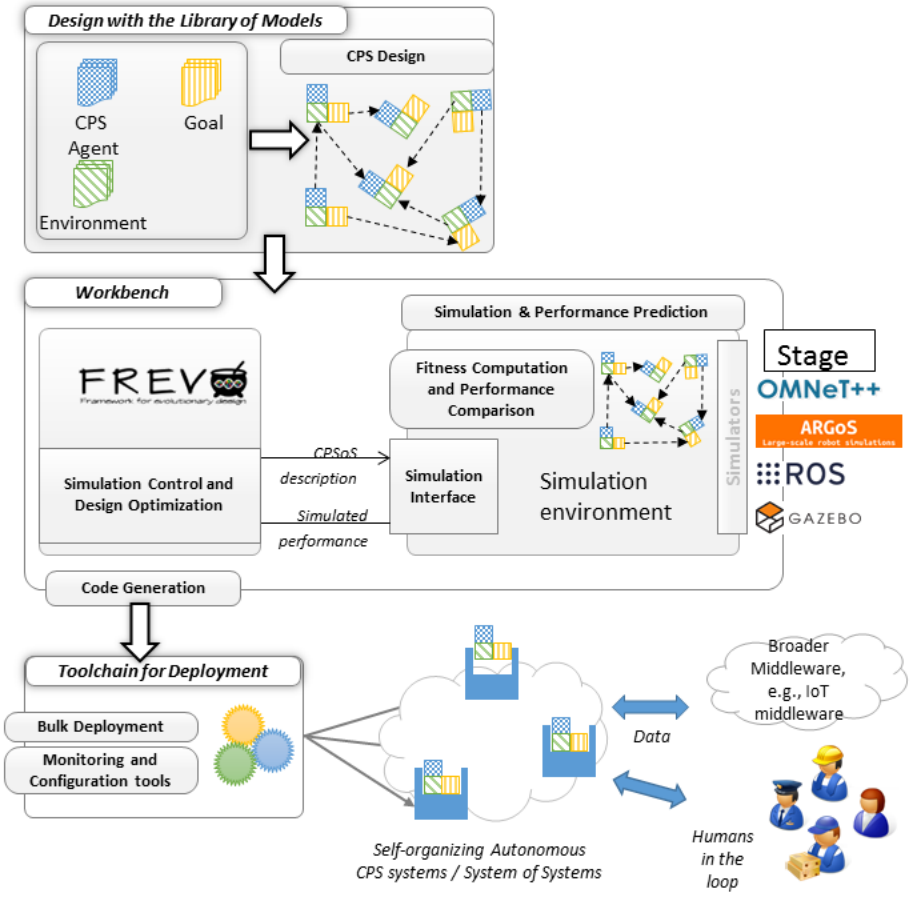

Figure 2: The CPSwarm high-level architecture and concept of designing swarm of CPSs

CPS designs may be optimized using several swarm intelligence algorithms such as ant colony, fish swarm and particle swarm optimization. On the other hand, evolutionary optimization can be used for designs which require an optimum solution in an uncertain environment. The CPSwarm project supports evolutionary optimization by directly integrating FREVO into the CPSwarm Workbench, see Section 3.3.

3.2 Evolutionary optimization

Designing swarms of CPSs poses two main challenges. First, selecting the hardware that best suits the requirements of the swarm, and second, designing the control algorithm defining the behaviour of the individual swarm agents. Approaches for designing the local controllers of a swarm of CPSs, or more generally self-organizing systems, can be categorized into two groups: hierarchical top-down designs starting from the desired global behaviour of the swarm and bottom-up designs based on defining the swarm agents and observing the resulting global behaviour [4]. The design using either approach is still a difficult process as neither can predict the resulting swarm behaviour based on the complex interactions between the agents [5]. This is especially true in dynamic environments. Evolutionary methods can be used to tackle such design challenges.

Designing a swarm of CPSs by using evolution is mostly an automatic design method that creates an intended swarm behaviour as a result of a bottom up process starting from interactions between very small components. The process gradually modifies potential solutions until a satisfying result is achieved. Such an evolutionary design approach is based on evolutionary computation techniques and can be done either on individual or on a swarm level.

Typically, the process of evolving a behaviour starts with the generation of a random population of individual behaviours. Each of these individual swarm-level behaviours is evaluated through simulation and ranked by computing a fitness function. Nevertheless, designing by evolution poses several challenges, including no guaranteed, predictable convergence, complex data structures and the high costs of evolutionary computation itself.

Furthermore, while designing a swarm of CPSs, a solution refers to a control algorithm of individual agents that is gradually improved during the optimization process. As experiments with real hardware require an extensive amount of time, such methods typically employ accurate and fast simulations to evaluate the performance of candidate solutions in the evolutionary process. The evaluation of algorithms in evolutionary optimization can be easily executed in parallel, which is for example supported in the FRamework for EVOlutionary design (FREVO), by using multiple cores on the same machine or by distributing the evolutionary optimization with a client-server protocol [6].

In summary, designing by evolution can be used to tackle challenges such as scalability and generality, as well as adaptive self-organization. Neither of these two issues is easy to handle, especially in changing environments and with dynamic interactions among individual agents of a system or a swarm. According to Fehervari and Elmenreich [7], there are six basic tasks related to evolution which designers must face while constructing a system model (Figure 3):

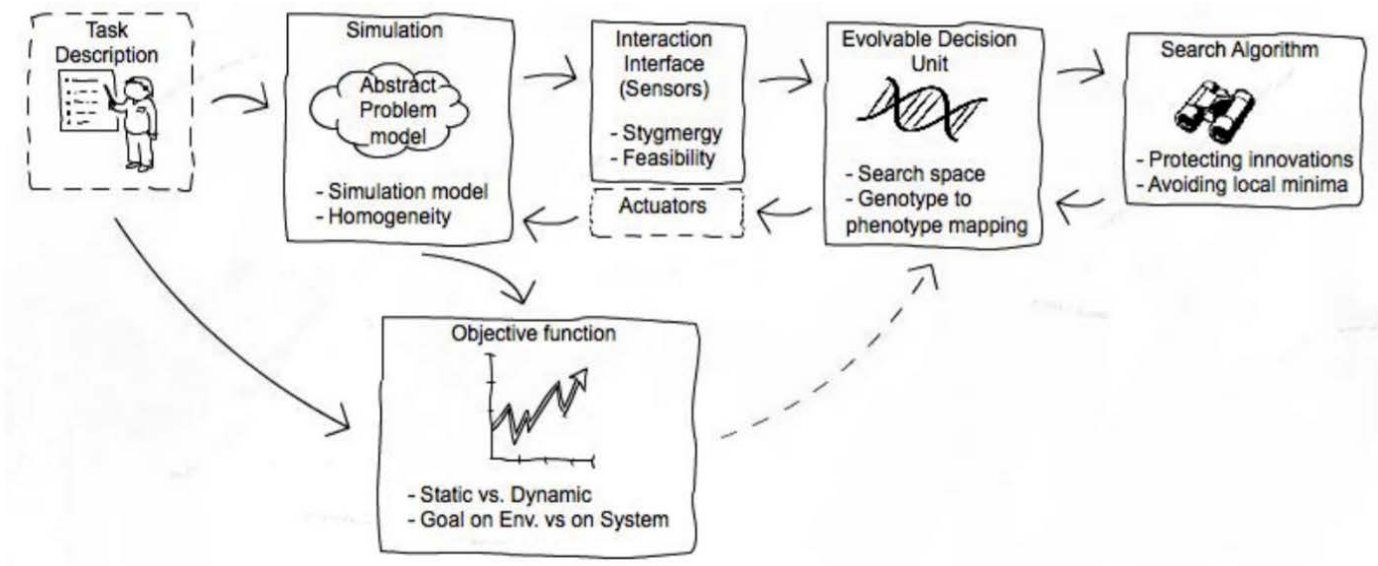

Figure 3: Evolutionary design methodology

The problem or task description gives a highly abstracted vision of the problem. This includes constraints and the desired objectives for such a problem.
The simulation setup transfers the problem description into an abstracted problem model.
The interaction interface defines the interactions among agents and their interactions with the environment.
The evolvable decision unit represents the agent controller and is responsible for achieving the desired objectives, i.e., the global behaviour of a swarm to achieve a common goal.
The search algorithm performs the optimization using evolutionary algorithms by applying the results from the above steps.
The fitness function represents the quality of the optimization result in a numerical way. It is highly dependent on the problem description.

3.3 FREVO as an optimization tool

FREVO is an open-source framework for evolutionary design or optimization tasks. According to the CPSwarm Workbench design, presented in D5.2, FREVO is integrated directly into the CPSwarm Workbench as an optimization tool and receives a complete set of modelling details for a problem from the Modelling Tool.

FREVO automates the problem setup phase, makes it straightforward to define local intelligence and interactions, and performs a search for best solutions that fit with genetic algorithms. Basically, there are many evolutionary and representation methods in FREVO, documented in [8], such as NNGA classic and CEA2D, which are genetic algorithms capable of evolving any kind of representation.

To develop a solution, FREVO needs an input consisting of several components as illustrated in Figure 4. First, it is necessary to define the problem where the evaluation context of the agent must be implemented. Second, a controller representation should be selected that describes the structure of a possible solution. Third, the optimization method must be selected to optimize the chosen controller representation to maximize the fitness returned from the problem definition. Finally, the ranking module is configured to evaluate all agents in a problem and return a ranking of the candidates based on their fitness.

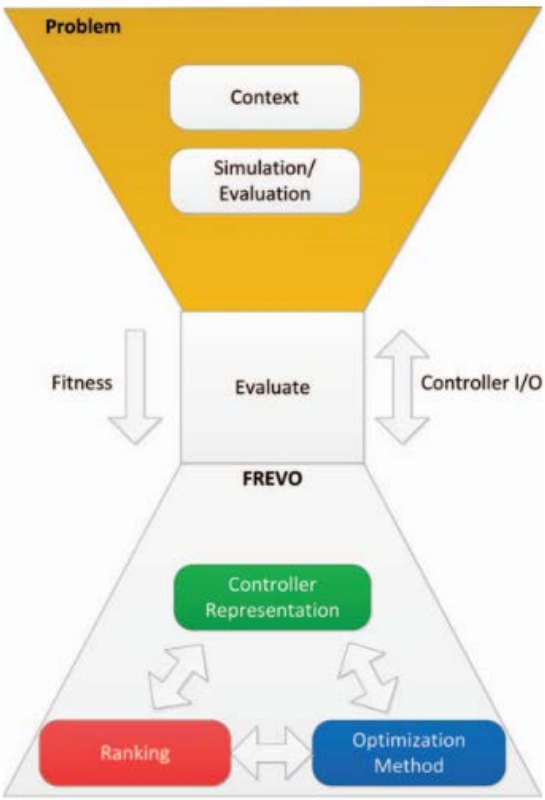

Figure 4: FREVO Architecture

4 Fitness function design

A fitness function assesses the behaviour of a swarm of CPSs: The best performance of the design optimization can be achieved by maximizing that fitness function. In literature, it has many names, such as fitness function, cost function, or utility function. Basically, it is a numerical representation that guides an optimization algorithm to find the best solution and it can be also a result of a continuous monitoring of mission parameters during a simulation run. It is highly dependent on the problem in question and thus there is no straightforward way or rule to be followed while designing a fitness function. It is a critical part of an optimization process as its effectiveness is directly related to the effectiveness of this function. For example, in the SAR scenario, see Section 5, a fitness function can be defined based on the search time or success probability.

4.1 Main properties

A fitness function that is designed to reward a desired behaviour of swarm of CPSs is highly problem-dependent. Nevertheless, many studies in the field of evolutionary optimization have considered generic methods for fitness function design. In general, these methods may be categorized into a three-dimensional fitness space [9, 10]:

1- ***Functional vs. behavioural:*** A functional fitness is based on components that directly measure the way in which the system functions. A behavioural one rewards the system for displaying a given behaviour.
2- ***Global vs. local:*** Global fitness rewards the system based on information that is available to an external observer, while the local one is restricted to information available to a single component.
3- ***Explicit vs. implicit:*** An explicit function rewards the way in which a certain goal is achieved, while implicit fitness is focused on how much the goal is reached (e.g. a distance). Implicit functions are also extensively used in search algorithms operating in the behavioural space.

4.2 Fitness function design guidelines

In general, the applicability and performance of a fitness function depends on the employed optimizer, thus there are no universally suitable fitness functions [11]. However, in the scope of the CPSwarm Workbench, we have identified the following guidelines for defining working fitness functions:

Defining scope and modelling sub-problems for complex goals: As mentioned above, a fitness function for a problem is directly related to the specifications for that problem. Nevertheless, a good start for designing an effective fitness function is to define the specifications for a given problem. Moreover, if an objective proves to be too difficult for a system, it might help to decompose it into simpler sub-objectives with lower utility values, for example, to evolve robots to play soccer it is good to reward players for kicking the ball since it directly correlates to the number of goals and consequently to the fitness of the solution [10].

Topology of fitness landscape: In general, adversary fitness functions [11], fitness functions with a large stochastic component (noise) and fitness functions with local cost minima can affect the optimization time and quality of optimization outcome. While the fitness is initially derived by the problem description, a refinement of the fitness function towards a "smooth topology" can significantly improve the result. Furthermore, the search space can be reduced by assigning high penalties towards unwanted behaviours (an example is a robot car that should go forward and orient itself, in this case, going backward could be excluded as behaviour). However, keep in mind that excluding certain behaviours might accidently cut off solutions which are not obvious but have superior performance in the end.

Combined fitness functions: In many cases the fitness function is comprised of orthogonal goals, for example, a robot swarm could be required to stay together, while having a second goal to move forward as a swarm. Typically, these goals can be easily expressed as separate fitness functions but not easily into a single

combined one. Some optimization algorithms can perform a multi-dimensional search which yields results in form of a set of non-dominated solutions. After all, this requires a selection based on a combined fitness before deployment and not all optimization algorithms in CPSwarm support this approach. Therefore, fitness functions are often described as a weighted sum of criteria, which shifts the problem to defining proper weights. While this ultimately depends on initial requirements, a quick guideline can be provided to normalize criteria based on their measured variance in order to get a set of equally matched criteria.

Computational effort to derive the fitness value: In some cases where the fitness function is derived from a simulation of the target system, the computational effort for computing the fitness function can become the defining part of the overall evolutionary algorithm. In some cases, typically early in the optimization process, a simpler fitness calculation which is considerably faster could significantly speed up the process. For the example where a fitness function is derived by a simulation this could be done with fewer repetitions of simulations (e.g. averaging the results of a few simulations in the beginning and increasing this amount at later generations, where accuracy is needed), shorter simulation time (adjusting simulated time depending on generations) or reduced accuracy (simulating with larger time steps/lower resolution in early generations).

4.3 Fitness function design in the CPSwarm Workbench

As indicated in Section 4.2, the design of the fitness function must follow the definition of the problem. For this reason, during the modelling phase, the CPSwarm Workbench allows a fitness function to be associated with a behaviour, after that the same Modelling Tool (i.e. Modelio[1]) has been used to define the state machine to address the problem. Furthermore, as shown in the SAR scenario in the next section, the state machine approach may be used to divide the complex scenario into several simpler states (e.g. the SAR scenario is subdivided into Coverage, Tracking, and Find Exit states) and then a separate fitness function can be designed for each state.

As indicated in the guidelines for designing the fitness function, the CPSwarm Workbench allows high penalties to be assigned to undesirable behaviours to reduce the time required to find the optimal result. Furthermore, it allows the optimization of the algorithm to be executed in two different steps: Firstly, the user can design a simple fitness function to be applied on a large set of candidates. Then, leveraging the initial results, a more complex fitness function can be applied to a restricted set of candidates, obtaining fine grained results. Typically, the fitness function is designed in the Modelling Tool, which allows the fitness function to be described using mathematical expressions, such as those presented in Section 5.3. When the fitness function has been modelled, the Modelling Tool uses this model to generate the code that will then be used in the rest of the Workbench.
The model of the fitness function is also stored in the Modelling Library, allowing it to be reused in different contexts. Furthermore, the library is pre-loaded with a set of default fitness functions associated with the behaviours designed for the proposed scenarios. For example, in the SAR scenario, the library could be preloaded with default fitness functions for all the possible states:

- To evaluate the *Coverage* algorithm, a fixed number n of checkpoints could be distributed over the area to be patrolled. In this case, the fitness function could count the number of checkpoints a drone flies within a distance d within a time interval t (other examples are presented in Section 5.1).
- To evaluate the *Tracking* algorithm, the fitness function can consider the proportion of time in which the drone is within distance d of the objective.
- For the *Find Exit state*, the value to be evaluated can be the average time required to escort all targets to an exit within a maximum allowed time t.

[1] https://www.modelio.org/

It is important to note that all the fitness functions - described through mathematical expressions - are reusable in different contexts by referencing parameters/properties of the model. For the example described in Section 5.1, during the modelling phase in Modelio, the user can decide what fitness function to apply to the *Coverage* state optimization, for example the *Coverage* function, which measures the number of visited fields within a given timeout. After selecting this fitness function, the designer can set its parameters (i.e. the value of the timeout, the size of the arena or the grid map).

The CPSwarm Workbench has a distributed architecture for the Simulation and Optimization Environment (see deliverable D3.2 – Updated system architecture & design specification, for the description of the different components), with several simulation engines distributed on different machines, each wrapped by a Simulation Manager. The distributed simulators are coordinated by a centralized component, called the Simulation and Optimization Orchestrator (SOO), which handles their interaction with the Optimization Tool (i.e. FREVO).
When the user has finished designing the fitness function and setting the desired values for the parameters, Modelio generates the related code, which is passed to the SOO through the Launcher.

The SOO chooses appropriate simulator instances for the optimization process and sends the models and the code of the fitness function to the related Simulation Managers. The SOO then instructs the Optimization Tool to start the optimization process and to use the configured simulation engines for the simulation. Following this, the Optimization Tool sends the candidates to be evaluated to the Simulation Managers. These execute the simulations and use the fitness function code to evaluate the results of each simulation and to calculate the fitness score of the candidate to be returned to the Optimization Tool, which uses these values to evolve the algorithm.

5 Case study on the Search and Rescue Scenario

5.1 An example Search and Rescue scenario (simulated environment)

A prominent example for the demonstration of swarms of CPSs is the SAR scenario. In such a scenario, multiple agents are deployed in an a-priori unknown environment to search for victims (referred to as targets) and to support their rescue.

In the CPSwarm project, the SAR scenario is one of the main vision scenarios used for demonstrating the Workbench (see deliverables D2.1 and D2.2). In this scenario, a swarm of drones searches a given area for targets and tracks found targets, supported by a swarm of ground rovers that guides the found targets to a safe place. This swarm behaviour may be formalized using state machines, as shown in Figure 5. For more details refer to future deliverable D4.5.

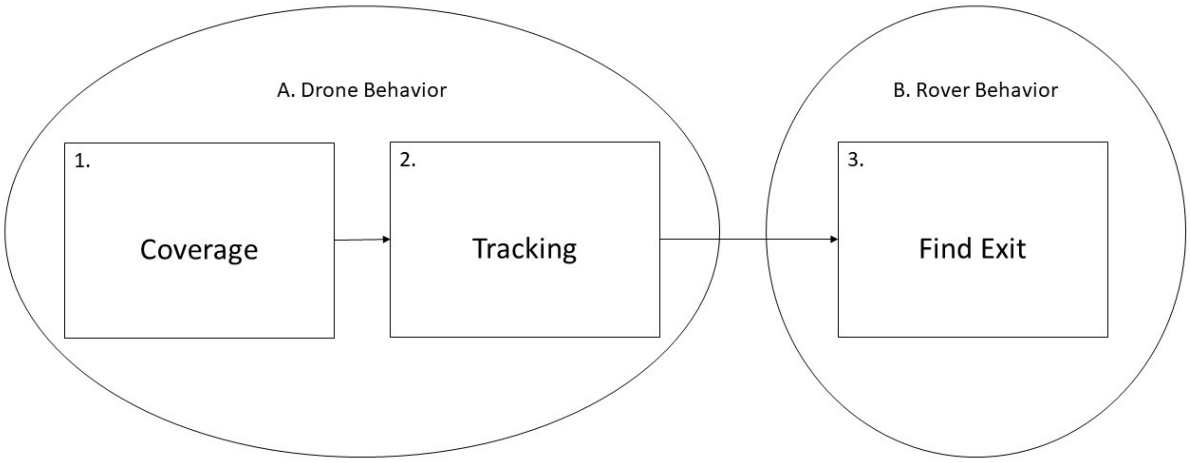

Figure 5: High-level states of the CPSs in the Search and Rescue scenario.

For each state of this state machine, different algorithms can be employed. These can be either bio-inspired swarm intelligence algorithms or algorithms that are evolved with evolutionary optimization. In this case study, we choose the coverage state to be evolved for further analysis and examine the influence of different fitness functions on the optimization process.

5.2 Coverage with a Swarm of Drones

For evolutionary optimization, we must complete the six tasks mentioned in Section 3.2.

1. Task Description
   In the coverage state, multiple drones fly over a given environment to localize the targets. The drones work together in a swarm to minimize the time until the targets are found. Ideally, the environment is covered completely by the drones while minimizing the overlap between the areas covered by each drone.

2. Simulation
   The coverage task is simulated with a given number n of drones that can move in discrete time and discrete space. The simulation always runs until a given maximum time t is reached. The environment is a rectangular, two-dimensional grid consisting of w horizontal and h vertical cells, yielding an area of $w \cdot h$ grid cells. It is assumed that the drones fly in one plane over the environment of interest. Cells can either be occupied by a drone, free, or occupied by an obstacle. Obstacles are placed randomly and occupy a given percentage o of the environment. In each time step a drone can move

to one of the four neighbouring fields if they are free. For a grid cell to be covered, at least one drone must visit that cell during the simulation.

3. Interaction Interface
   Each drone can sense whether neighbouring cells are occupied and differentiate between obstacles and drones. Obstacles are detected in the von Neumann neighbourhood consisting of the four closest cells. Drones are detected in the Moore neighbourhood consisting of the eight closest cells. This results in 12 sensor values as input to the decision unit. Each drone can move to one of the cells in the von Neumann neighbourhood. The direction to move in is given as x and y offset. Therefore, there are two actuator commands returned by the decision unit.

4. Decision Unit
   The decision unit is a time-discrete, recurrent artificial neural network (ANN) with one hidden layer, resulting in three layers total. The ANN is fully meshed meaning every neuron is connected to every other neuron and itself.

5. Search Algorithm
   The search algorithm is a neural network genetic algorithm (NNGA) that supports multiple populations and different selection schemes while trying to maximize the population diversity. The evolutionary search is performed for 100 generations where each generation has a population of 200 candidate solutions.

6. Fitness Function
   The fitness function measures the performance of the swarm. Regarding the coverage task, there are many parameters that can be used to construct a fitness function. A selection of some parameters is given below.

   - Swarm size n.
   - Number of time steps t in the simulation.
   - Size of the environment: Width w, height h.
   - Grid map of the environment:

$$M = \{m_{x,y} | x \in [0, w), y \in [0, h)\}$$
$$m_{x,y} = \begin{cases} 0, & \text{if cell } (x, y) \text{is free} \\ 1, & \text{if cell} (x, y) \text{is occupied} \end{cases}$$

   - Number of visits to each cell of the grid map:

$$V = \{v_{x,y} | x \in [0, w), y \in [0, h)\}$$
$$v_{x,y} \in [0, n \times t]$$

   All the parameters require global observation of the swarm. They can be used to implicitly measure the behaviour of the swarm, see Section 4.1. Using these parameters, different fitness functions can be constructed:

   - Coverage: Percentage of visited fields after given timeout
   - Redundancy: Number of times that the fields have been visited
   - Actuality: Maximum time between two visits of a field
   - Duration: Time until everything is covered
   - Activity: Percentage of agents working simultaneously
   - Distribution: Spatial distribution of agents

5.3 Fitness Function Comparison

Two of the fitness functions mentioned in the previous section have been implemented and evaluated. These are the coverage function, called C and the redundancy function, called R. They are formalized in the following way.

$$C = \frac{\sum_{x=0}^{w}\sum_{y=0}^{h}\left\lceil\frac{V_{x,y}}{n \cdot t}\right\rceil}{w \cdot h - \sum_{x=0}^{w}\sum_{y=0}^{h} M_{x,y}}$$

$$R = \frac{\sum_{x=0}^{w}\sum_{y=0}^{h}\left\lceil\frac{V_{x,y}}{n \cdot t}\right\rceil \cdot \left(1 - \frac{V_{x,y}-1}{n \cdot t}\right)}{w \cdot h - \sum_{x=0}^{w}\sum_{y=0}^{h} M_{x,y}}$$

To compare the fitness functions quantitatively, the following performance metrics are proposed:

- Complexity:
  - How computationally complex is it to compute the fitness?
- Convergence:
  - Does the fitness converge?
  - What is the maximum fitness reached?
  - How fast does the fitness converge?
- Sensitivity:
  - How do parameter changes influence the convergence?

Both fitness functions have the same **complexity**

$$\mathcal{O}(w \times h)$$

which is due to the fact that both iterate the environment grid map and the visited grid once. Therefore, no advantage can be seen for either fitness function from the complexity analysis.

The optimization process is executed for a number of different parameter settings. All possible parameter values are given in Table 1.
. Simulations were carried out for all parameter permutations except simulation time and environment size that were coupled such that $t = w^2 = h^2$ holds at all time. This yields a total of 300 different setups. To accurately measure the fitness of the simulations with both fitness functions, the optimization for each setup is repeated 25 times with varying random number generator seed. This results in a total of 7500 optimizations.

Table 1: Parameter settings for evolutionary optimization

Fitness function f	C, R
Simulation time t	100, 400, 900, 1600, 2500
Environment size $w = h$	10, 20, 30, 40, 50
Swarm size n	1, 2, 4, 8, 16
Obstacle percentage o	0, 5, 10, 15, 20, 25

The **convergence** behaviour of each fitness function during optimization is shown in Figure 6. Both converge with only minor differences. The coverage function is on average 0.9 percentage points better than the redundancy function. This is due to the fact that both functions compute coverage similarly but the redundancy function is penalized for redundant visits.

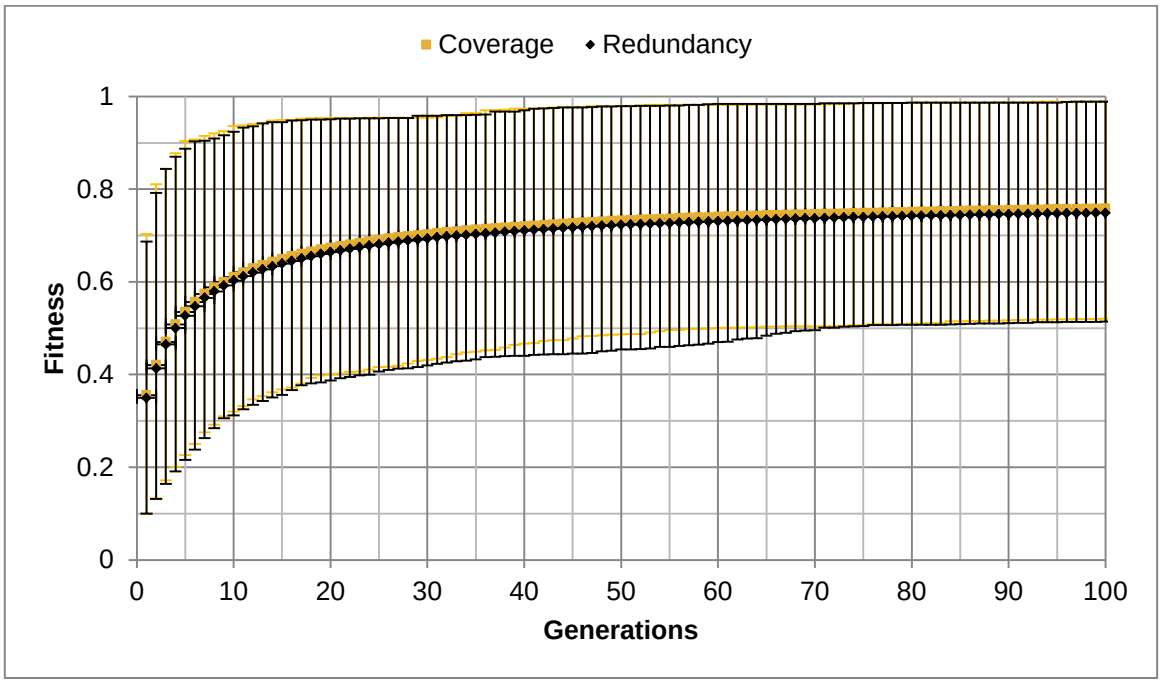

Figure 6: Average fitness with 10% and 90% percentiles for both fitness functions.

The convergence is quantified in Table 2, indicating what value the fitness converges to and how fast.

Table 2: Convergence of average fitness for different functions

	Coverage	Redundancy
Maximum fitness	75.83 %	74.90 %
Generations to reach 90% of maximum fitness	23	23

As shown above, the coverage function has a slight advantage over the redundancy function.

A **sensitivity** analysis was performed to determine the robustness of the fitness functions to parameter changes. Figure 7 shows an example of the two fitness functions for a different parameter setup. It can be seen that the setup affects both fitness functions differently.

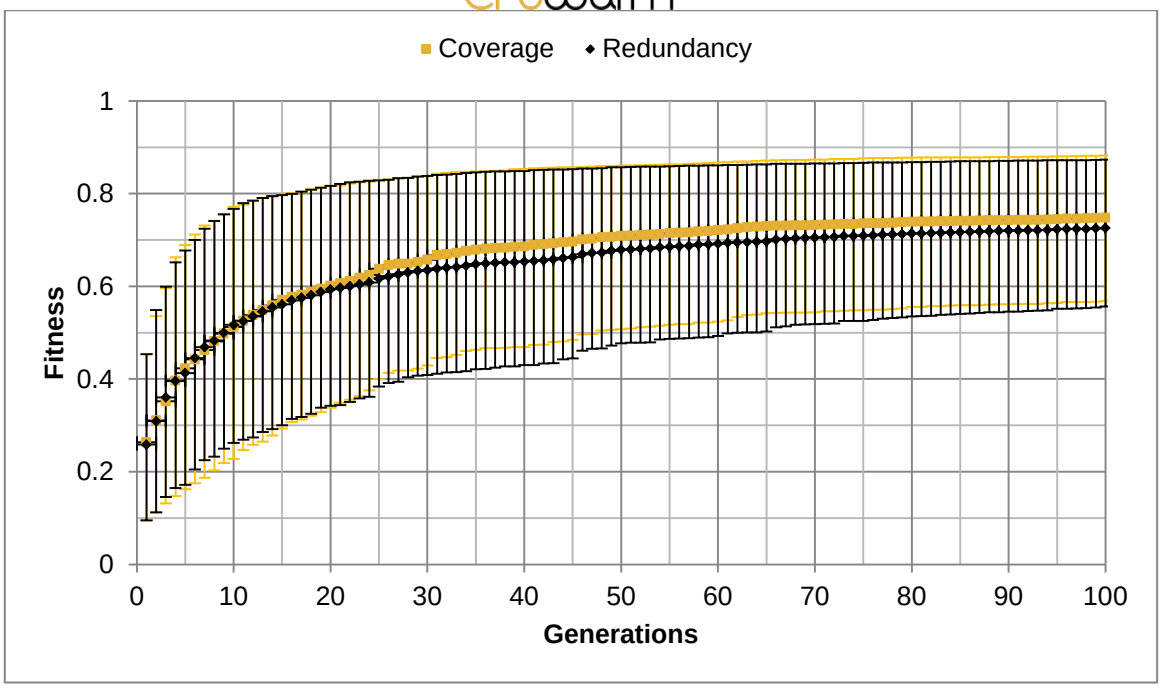

Figure 7: Average fitness with 10% and 90% percentiles for both fitness functions, varying swarm sizes, $t = 900$, $w = h = 30$, and $o = 10$.

Figures 8 to 13 show the influence of the parameters on the maximum fitness and on the speed of convergence, respectively. In both cases, the fitness of the optimization runs is averaged over the remaining parameters that are not mentioned. It can be seen that both fitness functions react similarly to parameter changes due to the fact that both functions are computed similarly. Nonetheless, in most cases the coverage function slightly outperforms the redundancy function.

Looking at the parameters swarm size n (Figure 10 and Figure 11) and obstacle percentage o (Figure 12 and Figure 13), the results show an increased performance for larger swarms and less obstacles. This can be explained trivially by the fact that with larger swarms more parallelization happens and with fewer obstacles the swarm members can move more freely. It can also be seen from Figure 13 that without any obstacles the fitness converges very fast and the evolutionary process does not need to be that long. It must be noted that the swarm size is still low enough to not create congestions in the environment.

A very interesting trend can be seen in Figure 9. The convergence speed of the optimization decreases with increasing environment sizes but only up to a certain size. After that, the optimization converges faster again. This suggests that small environments have a negative effect on the evolutionary optimization and should be avoided for the chosen fitness functions.

To conclude this comparison, we propose to use the coverage fitness function as it usually outperforms the redundancy fitness function while being simpler to compute.

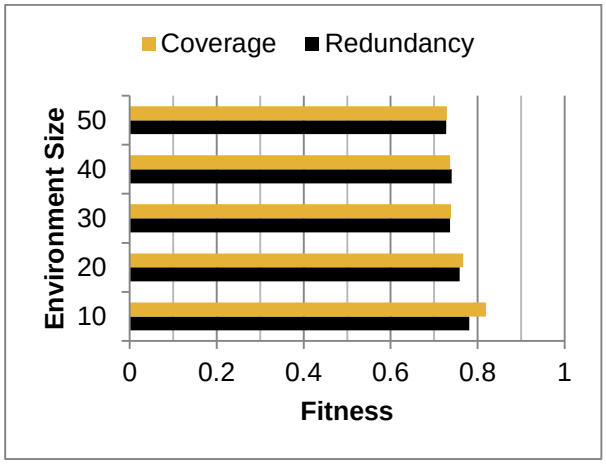

Figure 8: Maximum fitness reached on average for varying environment size $w = h$.

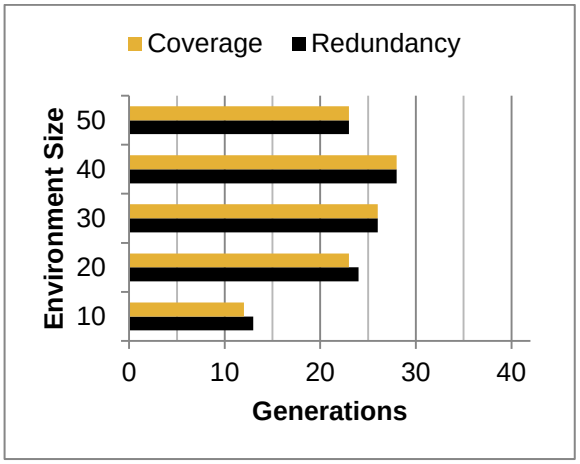

Figure 9: Generations to reach 90% of max. fitness for varying environment size $w = h$.

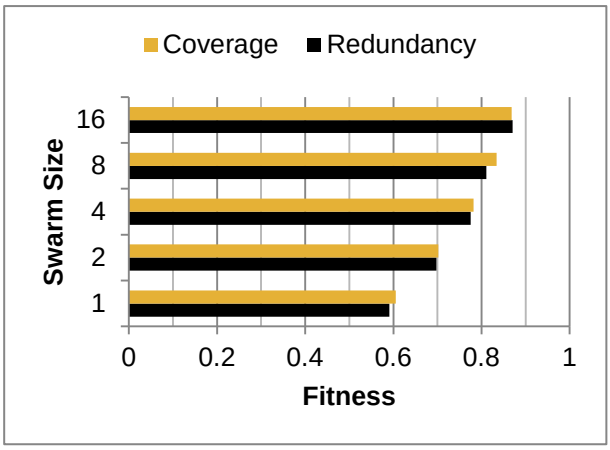

Figure 10: Maximum fitness reached on average for varying swarm size n.

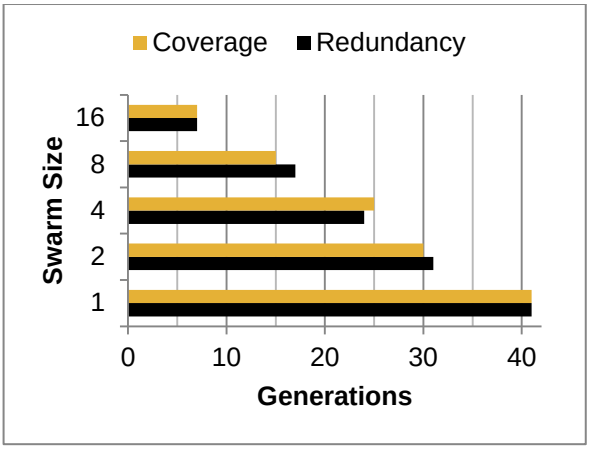

Figure 11: Generations to reach 90% of max. fitness for varying swarm size n.

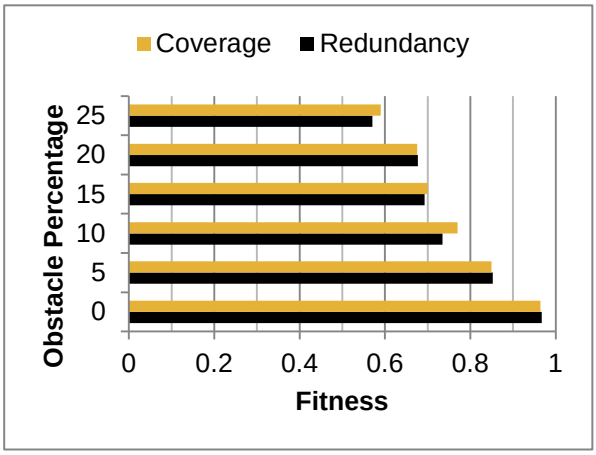

Figure 12: Maximum fitness reached on average for varying obstacle percentage o.

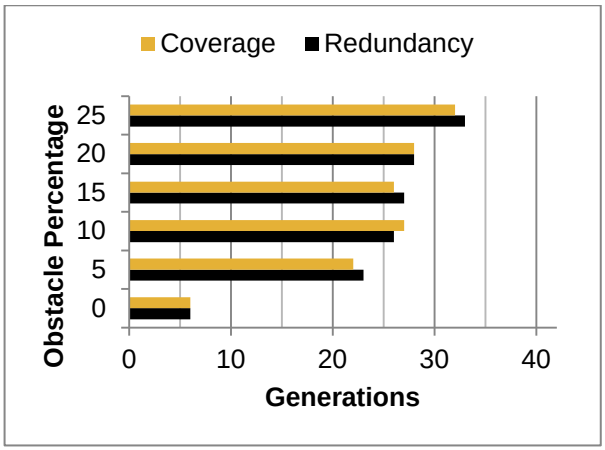

Figure 13: Generations to reach 90% of max. fitness for varying obstacle percentage o.

6 Conclusion

This deliverable describes the main steps for a swarm of CPSs design optimization using an evolutionary approach and provides initial guidelines for designing a fitness function that rewards desirable behaviour of a swarm of CPSs. The document proposes that the fitness function design for a system is highly dependent on the system specifications. Therefore, defining the specifications of a system or a problem will give hints for the design of an effective fitness function. The comparison of the effect of the different fitness function, for the coverage state using a swarm of drones, shows interesting results. For example, up to a certain size of the environment, the convergence speed of the optimization decreases with larger environments.

Future results and the progress of CPSwarm project with respect to evolution and fitness function design will be addressed in the Final CPS System design optimization and Fitness function design guidelines report.

Acronyms

Acronym	Explanation
CPSs	Cyber-Physical Systems
FREVO	FRamework for EVOlutionary design
SOEnvO	Simulation and Optimization Environment Orchestrator
SAR	Search and Rescue
ANN	Artificial Neural Network
NNGA	Neural Network Genetic Algorithm

List of figures

List of tables

References

[1] Schranz, Melanie, Alessandra Bagnato, Etienne Brosse, and Wilfried Elmenreich. "Modelling a CPS Swarm System: A Simple Case Study." (2018).

[2] Schranz, Melanie, Wilfried Elmenreich, and Micha Rappaport. "Designing Cyber-Physical Systems with Evolutionary Algorithms." In *Cyber-Physical Laboratories in Engineering and Science Education*, pp. 111-135. Springer (2018).

[3] Bagnato, Alessandra, Regina Krisztina Bíró, Dario Bonino, Claudio Pastrone, Wilfried Elmenreich, René Reiners, Melanie Schranz, and Edin Arnautovic. "Designing Swarms of Cyber-Physical Systems: the H2020 CPSwarm Project." In Proceedings of the Computing Frontiers Conference, pp. ACM, 2017.

[4] Crespi, Valentino, Aram Galstyan, and Kristina Lerman. "Top-down vs bottom-up methodologies in multi-agent system design." *Autonomous Robots* 24, no. 3 (2008): 303-313.

[5] Fehérvári, István, and Wilfried Elmenreich. "Evolving neural network controllers for a team of self-organizing robots." *Journal of Robotics* 2010 (2010).

[6] Rappaport, Micha, Melanie Schranz, Davide Conzon, Enrico Ferrera, Midhat Jdeed, and Wilfried Elmenreich. "Distributed Simulation for Evolutionary Design of Swarms of Cyber-Physical Systems."

[7] Fehérvári, István, and Wilfried Elmenreich. "Evolution as a tool to design self-organizing systems." In *International Workshop on Self-Organizing Systems*, pp. 139-144. Springer, Berlin, Heidelberg, 2013.

[8] Sobe, Anita, István Fehérvári, and Wilfried Elmenreich. "FREVO: A tool for evolving and evaluating self-organizing systems." In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE Sixth International Conference on*, pp. 105-110. IEEE, 2012.

[9] Floreano, Dario, and Joseba Urzelai. "Evolutionary robots with on-line self-organization and behavioral fitness." *Neural Networks* 13, no. 4-5 (2000): 431-443.

[10] Fehervari. I. "On Evolving Self-organizing Technical System". PhD thesis, Alpen-Adria-Universität Klagenfurt.(2013).

[11] Lockett, Alan J. 2015. Insights From Adversarial Fitness Functions. In Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15). ACM, New York, NY, USA, 25-39. DOI=http://dx.doi.org/10.1145/2725494.2725501