Development of a Reinforcement Learning Inspired Monte Carlo Tree Search Design Optimization Algorithm for Fixed-Wing VTOL UAV Propellers

Moritz Thiele[*] and Adrian Staudenmaier[†] and Swapan Madabhushi Venkata[‡] and Mirko Hornung[§]
Technical University of Munich, 80333 Munich, Germany

A novel Monte Carlo Tree Search Optimization Algorithm that is trained using a Reinforcement Learning approach is developed for the application to geometric design tasks. It is capable of evaluating design parameters and demonstrates the successful application of reinforcement learning strategies on a physics informed design optimization task. The algorithm is intended to be used for the parametric design of the optimal geometry of a propeller for Fixed-Wing VTOL UAV but is also applied to an aircraft design problem with ease.

I. Introduction

THE development of high performance fixed-wing VTOL UAV for remote sensing or transport applications is more and more driven by complex, mission-specific requirements. In an effort to enhance overall system performance VTOL UAV tend to merge hover- and cruise propulsion systems into one hybrid propulsive powertrain which requires propellers tailored to multiple flow states. The aerodynamic design of these highly efficient propellers poses challenges for traditional calculation and optimization algorithms in terms of complexity and processor time due to the high number of interdependent parameters. To tackle this task, new approaches have to be employed. The concept of Reinforcement Learning, traditionally used to develop decision strategies for robots or games, enables a fast and efficient way to train a geometrical optimization algorithm.

A. Problem setup and general approach

It is well understood, which geometrical properties thoroughly define a propeller geometry. These properties include the propeller radius, the detailed blade geometry in terms of planform and pitch as well as the radial airfoil shape distribution.

The physical properties of a propeller change in a deterministic way when the above parameters are modified which enables each parameters influence on the propeller performance to be drawn as a function over it's design space. Due to the propeller performance also being dependent on the local inflow conditions, which will change for each operational state, it is impossible to predict a perfect propeller geometry for all possible use cases and flight states of a UAV. This has to be taken into account when designing an optimal propeller as it can only be best suited for a dedicated operational state or a weighted combination of multiple clearly defined states.

The dependencies of the overall propeller performance on individual properties will most certainly change when exposed to a different flow regime which makes a prior knowledge of each parameter's influence on the propeller behavior not universally applicable but rather dependent on the state of operation.

The realm of machine learning provides a novel approach to predict the influence of all geometrical properties and its optimal values for a propeller. Reinforcement learning algorithms are capable to solve problems where prior knowledge is scarce and the data sets used for the learning process are generated on the run.

Thus, a novel optimization algorithm has been developed which exhibits very good scaling effects with the number of parameters considered. It defines an optimal propeller geometry while still maintaining a very detailed geometrical parameter set of the propeller with up to 20 to 40 parameters which poses significant difficulties for a wide range of traditional optimization strategies. The task of assigning values to all optimization parameters is performed based on a Monte Carlo Tree Search algorithm that is trained using a Reinforcement Learning approach. The algorithm is called "**MO**nte **CA**rlo **T**ree search" - "MOCAT" in this paper.

[*]Research Scientist, Institute of Aircraft Design, Boltzmannstr 15, 85748 Garching, moritz.thiele@tum.de.
[†]Masters' Graduate, Institute of Aircraft Design, Boltzmannstr 15, 85748 Garching, adrianstaud@t-online.de.
[‡]Graduate Student , Institute of Aircraft Design, Boltzmannstr 15, 85748 Garching, swapanmv@gmail.com.
[§]Professor , Institute of Aircraft Design, Boltzmannstr 15, 85748 Garching, mirko.hornung@tum.de.

This paper will address the development of the algorithm and the modifications to the classical Reinforcement Learning approach that are necessary in order to generate rewards for the feedback.

The defined algorithm is implemented in Matlab and will be tested and analyzed. Subsequently it will undergo a benchmarking process where the capabilities and performance of the optimization will be assessed in comparison to other commonly used geometry optimization approaches such as a modified genetic algorithm developed by Langer [1] and Particle Swarm Optimization.

II. Technical Approach

A. Reinforcement Learning Process

The core of a reinforcement learning algorithm is a basic Markov Decision Process (MDP) as shown in Fig. 1. It employs an agent residing in a state s_t performing an action a_t and thus interacting with the environment. This action will place the agent in a new state s_{t+1} and the agent will receive a reward r_{t+1} which indicates the quality of the state. [2]

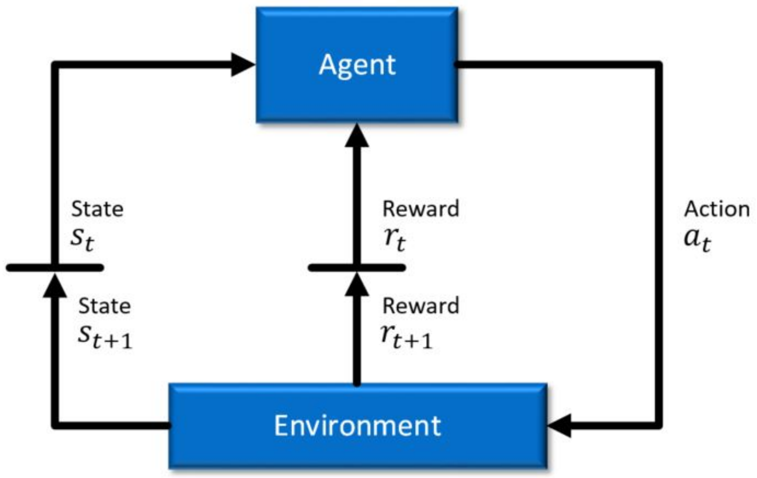


Fig. 1 Schematic structure of a reinforcement learning procedure.

1. Agent Description

An agent must be able to choose from a set of actions $\{a_i\}$ available in the current state s_t. The probability of choosing a certain action is determined by a state-value-function corresponding to an expected reward assigned to each possible action. As the number of available actions is large and can't be fully explored, the magnitude of the expected reward for each action has to be learned by reinforcement learning.

The decision which action to take will be made by evaluating a tendency of the agent for exploiting a high expected reward and for the random exploration of actions with an uncertain or lower expected reward. These tendencies for exploration and exploitation have to be adjusted dynamically during the run time of the algorithm.

When performing a geometry optimization, these actions do not represent individual steps performed by an agent such as the movement of a robotic part or the enactment of a move in a game, but rather assigning a distinct value to a geometrical parameter. The definition of an action for MOCAT is described below. Each parameter is defined by a lower and an upper boundary enclosing discrete or continuous values in between.

2. Environment and Reward Generation

The geometric optimization of a propeller will be performed for a given UAV mission. The UAV will fly in multiple flight phases for a specified duration during this design mission. These consist of vertical takeoff, hover, transition to forward flight, cruise- and loiter flight. Mounted on the UAV are several distinct propellers that are active during different flight phases. The geometrical optimization for a propeller has to strive for a weighted optimum of all its relevant flight states. One flight state is defined by the operational altitude, the UAV flight speed and attitude as well as external disturbances such as wind.

These environmental conditions are used to calculate a reward. It can be defined freely according to the design goal.

The reward that is needed for the algorithm is therefore generated by evaluating a target function which is represented by the propeller performance in these states and is calculated using an aerodynamic model that is able to provide sufficiently accurate results. Customary for a machine learning problem, the evaluation of a target function has to be conducted with many repetitions in order to ensure an adequately educated solution of the problem. [3]

Especially for a problem with many parameters, this requires the target function to compute a solution without manual input in a short time frame. Furthermore the target function needs to be sensitive to the geometrical parameters that are supposed to be optimized.

The aerodynamic model for the assessment of a single propeller in arbitrary inflow conditions is based on a modified Blade Element Momentum Theory (BEMT). The model is described in [4] and used here. The reward can be defined arbitrarily as a weighted combination of one or more performance values of the propeller for one or more flight states. A common performance metric for a propeller with an axial velocity $V > 0$ is the propulsive efficiency η, defined using the propeller Thrust and Power Coefficients C_T & C_P as:

$$\eta = J\frac{C_T}{C_P} \tag{1}$$

with J being the propeller Advance Ratio $J = V/(Dn)$, the inflow velocity divided by the propeller diameter and rotational speed in revolutions per second.

Studies for this publication have been conducted by using this propulsive efficiency of the propeller in one flight state directly as the reward: $r_t = \eta$.

Furthermore the environment can impose constraints on the design goal which can be defined freely and which have to be incorporated in the calculation. Two constraints that are used in this study are a limitation to the blade tip speed Mach number $Ma(V_{Tip})$ and a minimum thrust requirement T_{min} in order to fulfill a target mission.

B. Application to the Design Task

This concept will be applied to the design task of developing an optimal propeller. An explicit definition of the algorithm state and actions will be given after listing the relevant propeller optimization parameters.

1. Propeller Parameter Definition

A VTOL UAV propeller as sketched in Fig. 2 is usually a fixed-pitch propeller with two or more blades. A two bladed propeller will be used for the sample calculations in this paper.

The geometric set of parameters to be optimized is defined by global parameters such as the rotational speed Ω and the propeller diameter R as well as local parameters defined radially along the blades such as the radial distributions of the chordlength $c(r)$ and local pitch $\beta(r)$. Each parameter is defined by a lower and upper boundary enclosing discrete or continuous values in between.

The pitch and chordlength distribution of a propeller cannot be chosen freely along the blades as the rate of change along the radial coordinate is limited and must not oscillate. Thus these parameters are not considered directly at distinct radial positions but rather are derived from an envelope distribution that is shaped using design parameters.

A sample envelope for the chord and pitch distributions is shown in Fig. 3a and Fig. 3b. The parameters defining the silhouette are equal for both distributions and are highlighted in the figures. The shape is described as 3 consecutive parabolic arcs, each defined by two boundary points A, C, E and G where the arcs are interconnected as well as the peak locations B, D and F. The peaks are defined relative to the boundary points with two distances measured in relative percentage of the boundary point separation. Exemplary for the parabola $[E, F, G]$ these are defined as:

- Position M of the peak along the line between the boundary points $[E, G]$.
- Magnitude of the maximum perpendicular separation N between the peak F and the line $[E, G]$.

The boundary points are defined by their coordinate pairs (R, Y) with Y being the pitch or chordlength value respectively. A and G have a fixed radial coordinate of $r = 0$ and $r = R$ while all boundary points' Y Coordinates are limited by their upper and lower pitch or chordlength restriction. This representation decouples the discretization density used in the BEMT model for the performance calculation from the number of optimization parameters. The true values of the blade pitch and chordlength are read from the silhouette at the radial locations used for the discretization of the BEMT model.

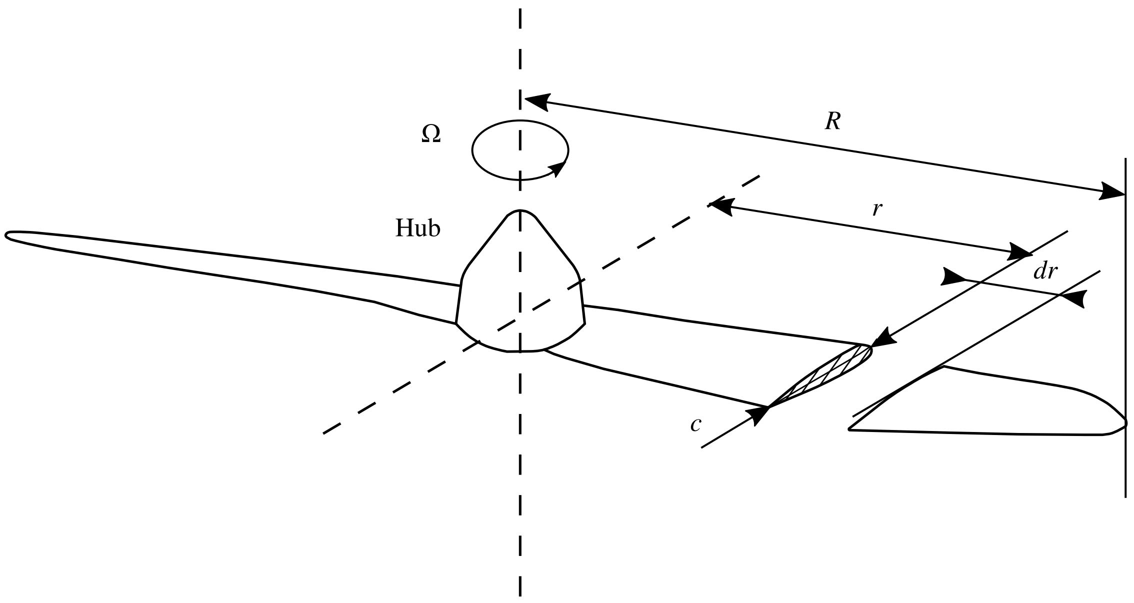
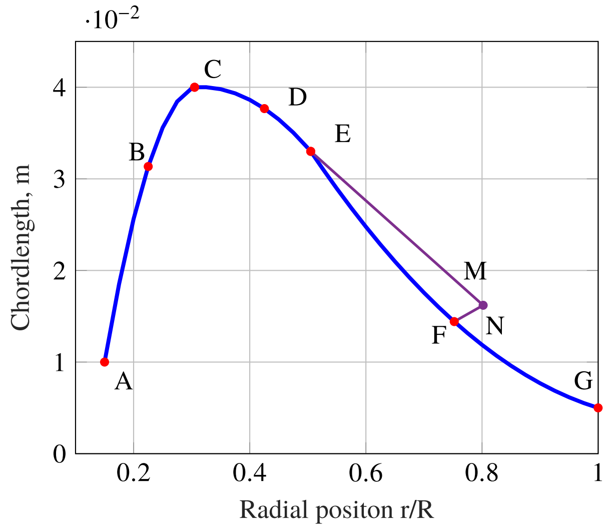
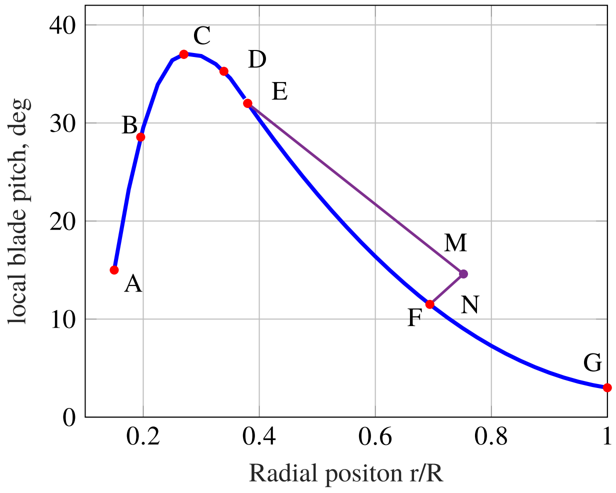


Fig. 2 Sketch of the propeller definition.



(a) Parameterized chordlength distribution.



(b) Parameterized pitch distribution.

Fig. 3 Representation of propeller pitch and chordlength design parameters

Table 1 Geometric propeller optimization parameters and boundary conditions

#	Parameter	Lower Limit	Upper Limit
1	R_{Tip} $[m]$	0.1	0.3
2	RPM $[1/min]$	1500	12000
3	R_{Hub} $[m]$	0.01	0.025
4	$Chord\ c\ Value\ (A)\ [m]$	0.003	0.024
5	$Chord\ c\ Value\ (C)\ [m]$	0.0156	0.045
6	$Chord\ c\ Value\ (E)\ [m]$	0.0072	0.0366
7	$Chord\ c\ Value\ (G)\ [m]$	0.003	0.0156
8	$Chord\ c\ Radial\ Position\ (C)\ [-]$	0.25	0.4
9	$Chord\ c\ Radial\ Position\ (E)\ [-]$	0.45	0.7
10	$Chord\ c\ Inner\ Peak\ Location\ (M_{ABC})\ [\%]$	0.35	0.65
11	$Chord\ c\ Mid\ Peak\ Location\ (M_{CDE})\ [\%]$	0.35	0.65
12	$Chord\ c\ Outer\ Peak\ Location\ (M_{EFG})\ [\%]$	0.35	0.65
13	$Chord\ c\ Inner\ Peak\ Distance\ (N_{ABC})\ [\%]$	-0.15	0.15
14	$Chord\ c\ Mid\ Peak\ Distance\ (N_{ABC})\ [\%]$	-0.075	0.075
15	$Chord\ c\ Outer\ Peak\ Distance\ (N_{ABC})\ [\%]$	-0.125	0.125
16	$Pitch\ \beta\ Value\ (A)\ [m]$	0.5	25.25
17	$Pitch\ \beta\ Value\ (C)\ [m]$	15.35	50
18	$Pitch\ \beta\ Value\ (E)\ [m]$	5.45	40.1
19	$Pitch\ \beta\ Value\ (G)\ [m]$	0.5	15.35
20	$Pitch\ \beta\ Radial\ Position\ (C)\ [-]$	0.25	0.4
21	$Pitch\ \beta\ Radial\ Position\ (E)\ [-]$	0.45	0.7
22	$Pitch\ \beta\ Inner\ Peak\ Location\ (M_{ABC})\ [\%]$	0.35	0.65
23	$Pitch\ \beta\ Mid\ Peak\ Location\ (M_{CDE})\ [\%]$	0.35	0.65
24	$Pitch\ \beta\ Outer\ Peak\ Location\ (M_{EFG})\ [\%]$	0.35	0.65
25	$Pitch\ \beta\ Inner\ Peak\ Distance\ (N_{ABC})\ [\%]$	-0.15	0.15
26	$Pitch\ \beta\ Mid\ Peak\ Distance\ (N_{ABC})\ [\%]$	-0.075	0.075
27	$Pitch\ \beta\ Outer\ Peak\ Distance\ (N_{ABC})\ [\%]$	-0.125	0.125

The propeller optimization parameters that are primarily used for this study as well as their respective boundary conditions are summarized in Table 1. The three parabolas defining the pitch and chordlength distributions are called "inner", "mid" and "outer" respectively. There are a total of 27 Parameters that define the propeller geometry and provide ample flexibility to represent multiple flow states including inclined inflow conditions.

One critical design parameter that is not included in the current set is the local airfoil shape. This algorithm requires all parameters to change the propeller behavior in a sufficiently continuous way over the values between the lower and upper limits. A list of airfoil shapes will change the propeller behavior in a non-deterministic manner. This behavior cannot be predicted by the current optimization algorithm without further improvements. Thus the calculations presented here use a default airfoil that is scaled with the local chordlength. The number of blades has been fixed to two as this is the most commonly used value for electrically driven UAVs and because the BEMT that is used to predict the reward is incapable of estimating the mutual influence between multiple blades which would lead to a tendency to use as many blades as possible during the design process. Another characteristic that is not represented in these sample calculations is the local blade sweep which is assumed to be zero. Including the sweep characteristic in the design optimization process is, however, easily possible.

2. State Definition

The main challenge of applying this algorithmic concept to a design task is exemplified when comparing the definition of the parameter set to the movement robotic agents - the more natural application of reinforcement learning algorithms.

A reward value for the current state of a robot can be calculated after the robot has performed one movement in the environment as this will place it in a new state. A robot will need to use its actuators to perform a certain amount of work in order to move. It is not possible to simply move one actuator of a complex robot to make a move. Similarly it is impossible to predict the performance of a design when only one of its design parameters has been assigned a distinct value. One action of the algorithm will need to assign exactly one distinct value to all of the design parameters in order to start the calculation of a reward value.

This leads to the following definitions of a state and an action when working with a design task: One state s_t consists of all parameters having exactly one value within their allowed range assigned. With this, the performance of the design can be evaluated and a reward can be generated. Subsequently one action a_t is performed in assigning a different value to each parameter thus placing the agent in a new state within the environment. This entails that the goal of the optimization is not a sequence of actions leading to a final state. Instead, the goal of the algorithm is finding the one optimal action to take which means assigning an optimal value to each parameter.

The parameters that have been defined in the previous section are structured in a search tree shown in Fig. 4 which is drawn with a reduced parameter count where each level represents one parameter P_j and each node n_i a value this parameter can obtain.

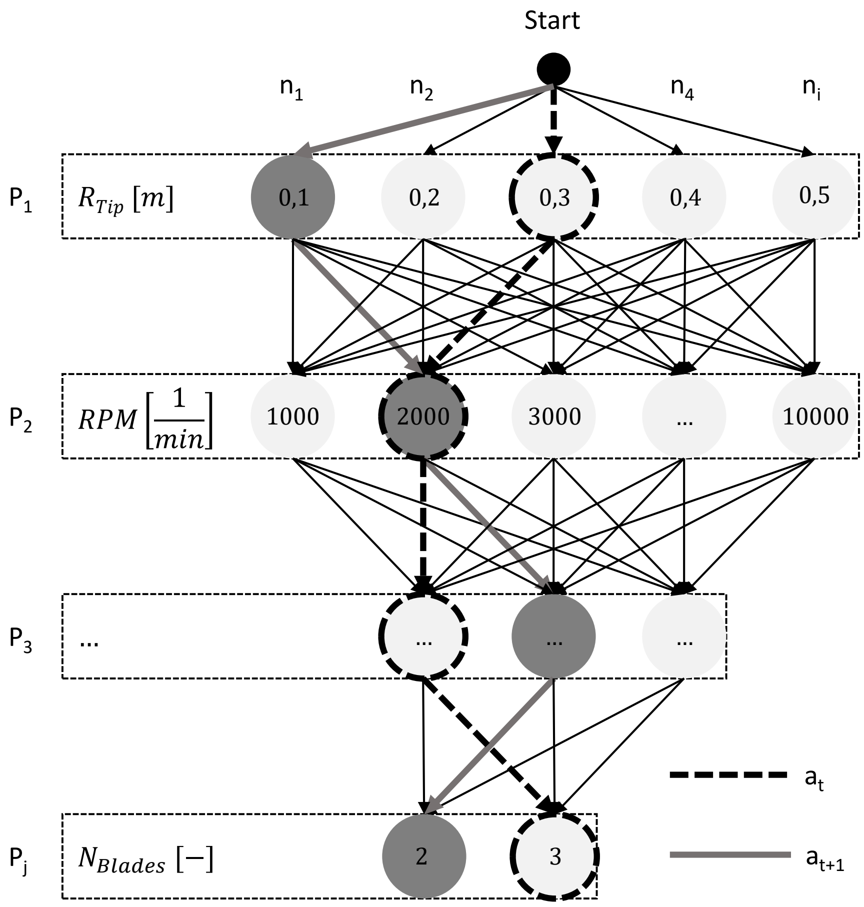


Fig. 4 Search tree with optimization parameters and two paths.

The nature of this search tree dictates that only discrete parameters are used for the optimization process. The number of nodes per parameter can be arbitrary, however, a large number of nodes will negatively affect the convergence speed of the algorithm. More than 15 to 20 nodes per parameter will reduce the convergence rate of the algorithm significantly and should be avoided. The number of different discrete values a parameter can obtain should therefore be small. On the other hand, it is necessary that parameters such as the chordlength can be chosen from a continuous parameter scope. This is done by virtually refining the resolution of nodes by a large factor for these continuous parameters which makes these parameters quasi-continuous during the parameter fixation procedure that is described in detail in the next chapter .

Each node of every parameter can theoretically be chosen independently of the nodes selected for other parameters enabling each node being connected to every node of the succeeding parameter. Furthermore the sequence of the levels in the tree can be arbitrary and can be subject to change while the depth of the tree is fixed and limited to the number of parameters. This will be important for the parameter fixation process described below. When searching for an optimal parameter set, the algorithm has to search for the path through the search tree that provides the best reward.

C. Parameter Assessment Process

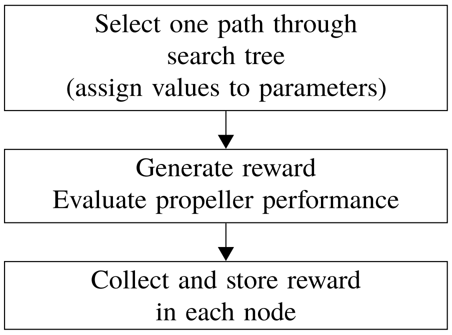

Select one path through search tree (assign values to parameters)
Generate reward Evaluate propeller performance
Collect and store reward in each node

Fig. 5 Event flow chart.

The final design values of the parameters have to be chosen within the respective scope and have to be fixed in order for the optimization to converge. This search for the best results is done in a multi layered approach that is inspired by the classical Monte Carlo Search Tree Expansion and Simulation practice used in [5].

The values assigned to the parameters in one action a_t will be picked according to a tendency of either exploration of the available nodes or exploitation of the most promising node. While exploring the parameter space, the path through the search tree chosen for one action will pass through a random node n_i of a certain parameter P_j. This path is shown in Fig. 4 by highlighting the visited nodes. The reward generated by this path is then stored in each node that is part of the path.

The next action a_{t+1} will create another path that is passing randomly through the nodes and will trigger a another reward being stored in these nodes. A node that is visited multiple times will thus get assigned multiple different reward values. This oft repeated process of choosing a path, evaluating the results and storing the reward simulates the impact of each node on the reward. It is depicted generically in Fig. 5 and is called event in the algorithm flowchart Fig. 9 below.

Calculating a weighted average of all rewards seen by a certain node n_i defines the magnitude V_i of a Value-Function $V_{P_j} = f(V_i)$ at this node of a Value-Function V_{P_j} created for each Parameter. This Value-Function, that is shown in Fig. 6a represents the expected reward for each node of a certain parameter P_j. It is represented by red dots for each node over the design space of each parameter. A good node that will maximize the reward is represented by a high Value-Function V_{P_j}. The magnitude of V_{P_j} is normalized and thus does not represent the reward directly. It is, however, used to evaluate the worth of the nodes of all parameters relative to each other.

The tendencies depicted by V will increase in accuracy as more actions are evaluated. When enough actions are stored, the algorithm will begin processing the results. As mentioned above, the resolution of nodes visited by a path is supposed to be quite coarse. Thus the resolution is then artificially enhanced in order to assess quasi-continuous parameters in a meaningful way. This is done as shown in Fig. 6b with the blue line by using a regression algorithm to predict the magnitude V_i on a continuous scale within the parameter boundaries. The regression algorithm will produce a large number of predicted nodes that have not been visited by the simulation but which can be chosen during the fixation process. The Value-Function will then individually be evaluated for each parameter by assessing the quality of the approximation and regression and by searching for the maximum value. This peak will then be rated by a peak clarity metric that can assess the informative value this peak holds with respect to the rest of the function-space. The process described here has the optimization goal of finding a maximum value. However, many optimization problems desire to minimize a target function. This can easily be achieved by inverting the Value-Function generated by the events prior to starting the assessment process described below thus enabling the search for the highest inverted V-Value. This is a valid approach as the assessment process only aims to search for the best location in the parameter space which is depicted on the X-axis of Fig. 7. Globally inverting the Value-Function will not change this best location but enables the assessment process to search for a maximum either way.

This process of assessing and rating the Value-Function is essential for the algorithm in order to choose parameters to be fixed and to select a certain node of most promising rewards. It will be explained in depth on the example of the parameter "RPM" shown in Fig. 7 that has been calculated using a lower resolution of visited nodes that it is shown in Figs. 6a and 6b.

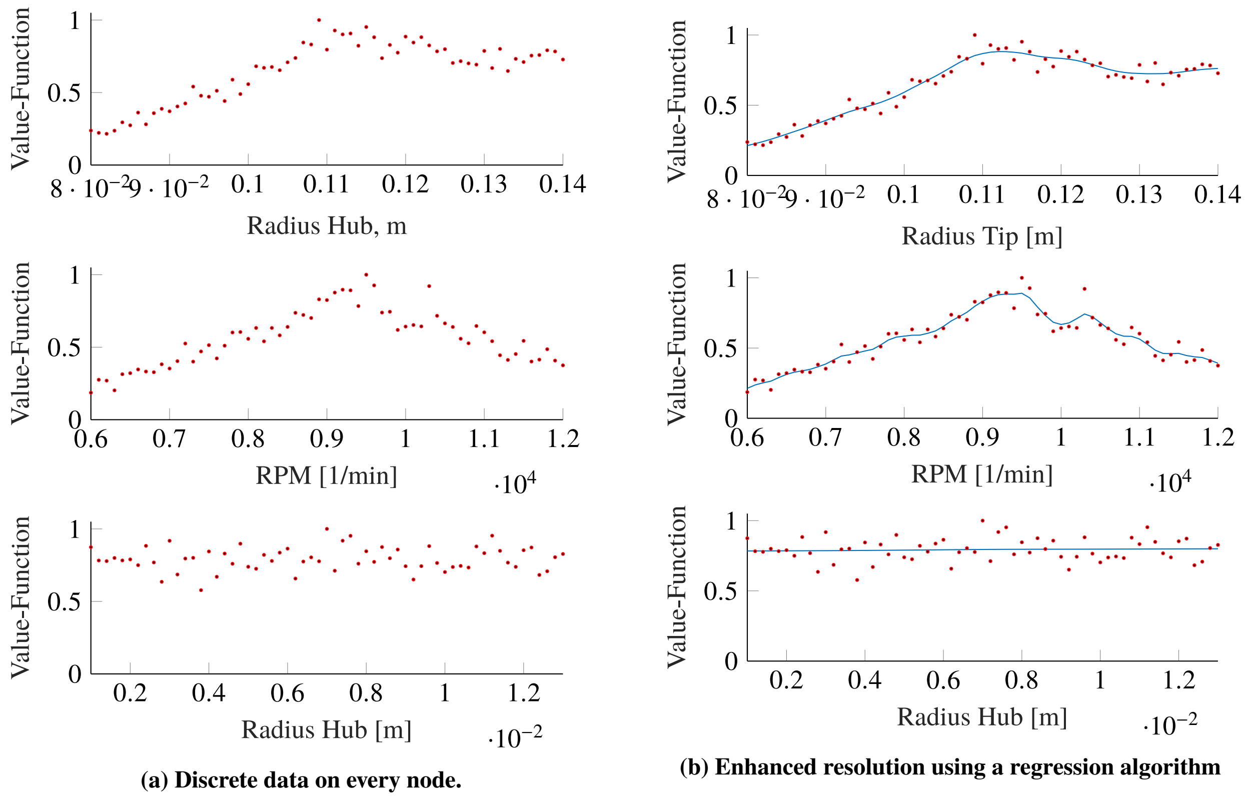


Fig. 6 Value function of three parameters.

1. Nodal Approximation Quality

The approximation quality in the design space is determined using a set of metrics that are aimed at quantifying the uncertainties in the results. They are evaluated for each visited node of a parameter depicted as a red dot in Fig. 7 and then combined to a single grade for each parameter.

First, a Validity Index is generated for every node that represents the ratio between successful and unsuccessful actions. An unsuccessful action can be triggered by the chosen path leading to the violation of a constraint imposed by the environment such as the blade tip mach number. Another category of unsuccessful events has to be dealt with when the reward evaluation function fails to converge and thus delivers no results. Unsuccessful evaluations are stored in the nodes with a reward of 0 which is the reason the Value-Function V has a quite low magnitude. This measure is important as a high probability of the action being unsuccessful indicates that a certain node has a high tendency to result in constraints being broken. The mean value of

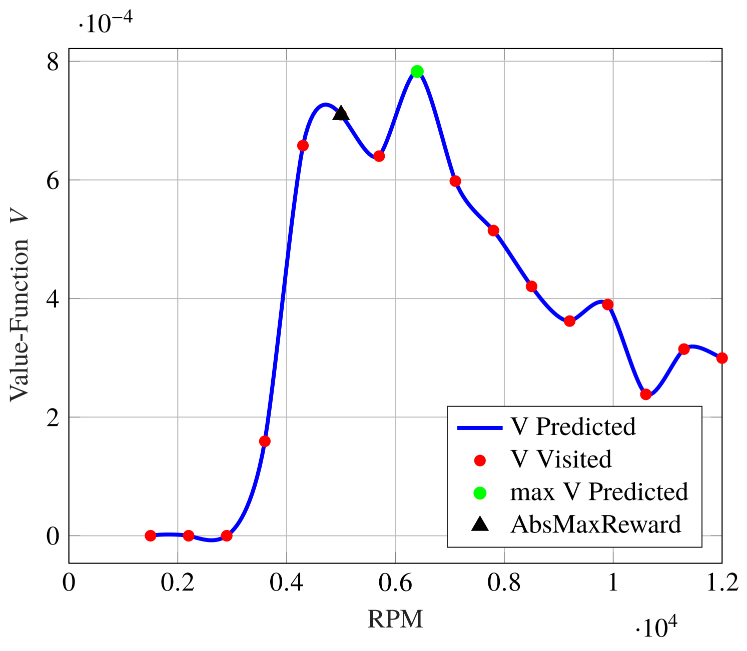
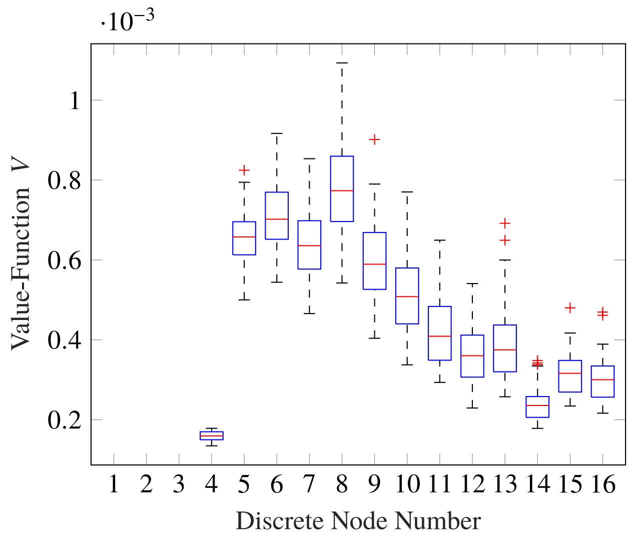
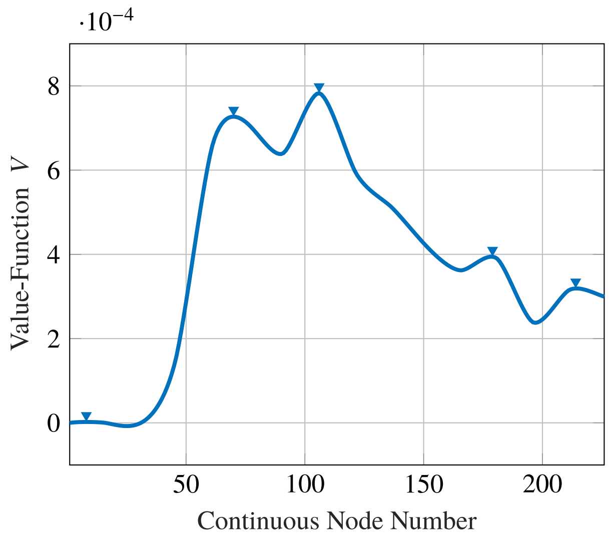


Fig. 7 Predicted reward of parameter RPM.



(a) Variance of V for visited nodes.



(b) Predicted peaks for RPM parameter.

Fig. 8 Information used for parameter assessment.

one parameter's validity indices is stored and used for comparison. The next two quality indicators that are assessed are shown in Fig. 8a. This plot depicts the reward variation between the successful actions for each visited node. The median reward of each node $\tilde{V}_i$ is shown as a red line as well as the first and third quartile - $Q_{25\%}$ and $Q_{75\%}$ - which are indicated by the blue box. The first two nodes have not been involved in a successful evaluation thus indicating a violation of an optimization constraint which can be seen in Fig. 7. The relative spacing between the two quartiles QS_{rel} for each node is calculated according to Eq. (2) and used as a quality indicator.

$$QS_{rel} = \frac{Q_{75\%} - Q_{25\%}}{\tilde{V}_i} \tag{2}$$

Similarly to the Validity Index, the ratio "$OutShare$" between successful actions and outliers depicted by red crosses in Fig. 8a is calculated. An outlier is defined to be more than three scaled median absolute deviations away from the median. The quartile spacing as well as the ratio of outliers indicates the impact of a certain node on the overall design. A low spread of rewards at a node corresponds to a large overall influence while a large spacing suggests the result being driven by other parameters.

The last components that are used to determine the approximation quality is the accuracy of the regression algorithm used to increase the continuous parameters' resolution. It is estimated by calculating R^2 and the root mean square error $RMSE$ of the regression line. A low R^2 and high $RMSE$ depict much discontinuity in V_{P_j} thus indicating the reward being driven by other parameters.

2. Peak Clarity Estimation

In order to assess the clarity of the peaks found in the Value-Function, it is necessary to first identify all present peaks as shown in Fig. 8b. A peak is identified by the blue line representing all predicted nodes decreasing on both sides. It is also possible for a peak to be present at the boundary of the parameter space if it is reached with an upward slope. This list of peaks will be sorted by their height for further use.

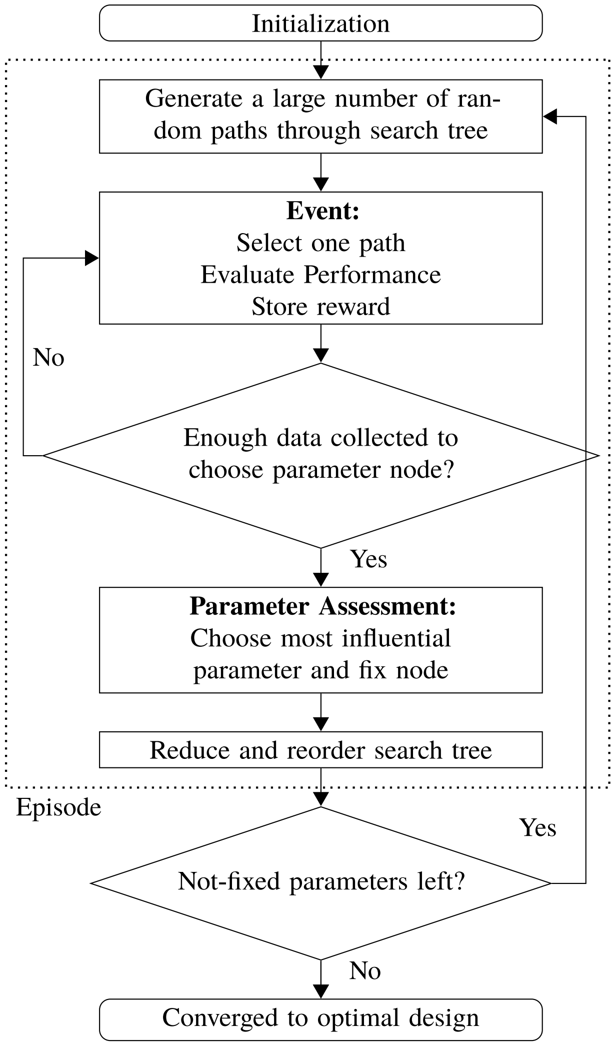
The simplest measure is the magnitude of V at the highest peak. Another important aspect is the peak separation factor PSF of every peak besides the highest peak that is computed as the horizontal distance divided by the vertical distance of every peak with respect to the highest one. The PSF depicts the prominence of the highest peak showing whether it is rather solitary or if there are other peaks in a different region of the parameter space with an almost equal height. A PSF number smaller than 1 indicates that the slope of the line connecting both peaks is steeper than 45 deg. In Fig. 8b, the PSF of the second highest Peak is greater than 1 which leads to the conclusion that a true maximum reward could be achieved somewhere between those peaks. The PSF of the third highest Peak is smaller than 1 indicating that this peak can be clearly seen as separated from the highest when searching for the best predicted reward.

Table 2 Parameter quality and clarity indicators.

	Indicator	Weight
Quality	Validity Index	7
	Quartile Spacing	8
	Outlier Share	5
	R^2	4
	$RMSE$	4
Clarity	Peak Magnitude	8.5
	Peak Separation Faktor	7
	Peak Width	5
	Peak Validity	5
	Peak Quartile Spacing	5
	AbsMax Location	2

Table 3 Final parameter assessment weights.

Nodal Approximation Quality	2
Peak Clarity	3



Initialization
Generate a large number of random paths through search tree
Event:
Select one path
Evaluate Performance
Store reward
No
Enough data collected to choose parameter node?
Yes
Parameter Assessment:
Choose most influential parameter and fix node
Reduce and reorder search tree
Episode
Yes
Not-fixed parameters left?
No
Converged to optimal design

Fig. 9 Algorithm flow chart.

The highest peak is furthermore characterized by its width. This width is measured horizontally relative to the overall parameter space and defined by the number of predicted nodes being less than 5% below the peak height. A wider peak indicates a strong tendency to depict a more global maximum while a narrow peak points to a more local maximum with the wider peak being preferable to a more narrow one in order to choose a stable configuration.

The certainty of the highest peak is measured by assessing the Validity Index and Quartile Spacing QS of the two nearest visited nodes. These being worse than the indicators' mean value over the whole parameter shows a high uncertainty at the peak location which reduces its informative value.

Lastly, the black triangle that is shown in Fig. 7 is incorporated in the assessment process. It depicts the location of the absolute maximum value encountered during the whole optimization procedure. It represents the result of a brute force optimization had it been conducted with all so far tried combinations. The distance between the highest peak and the black triangle is calculated and rated as this location might indicate a global optimum which would increase the clarity of the peak.

3. Parameter Assessment Summary

All indicators described above are transformed to a scale between 0 and 1 where 1 is the most preferred tendency. As a next step, they are weighted according to their relative importance. The weight values have been estimated based on a number of test runs of the algorithm and are summarized in Table 2.

The separate quality and clarity metrics are combined by another weighted sum as shown in Table 3 in order to create one comprehensive metric. This metric enables MOCAT to compare the different parameters' Value-Functions and to decide on the most promising parameter. The highest predicted peak of this parameters' design space will be chosen and fixed. A new node will be created in the tree structure of Fig. 4 if no node exists at this location when a quasi-continuous parameter is chosen.

D. Exploration vs Exploitation

As the target of the optimization is not a sequence of actions but rather finding exactly one optimal action, the exploitation of the knowledge gained has to be incorporated in the actions itself rather than the action sequence. All parameters' Value-Functions V and their predicted peaks will be compared and the most promising and influential candidate will be chosen.

The optimization process will then restart with the limitation that all paths run through this chosen node of the chosen parameter. This is represented by reordering the layers of the tree, that is moving the fixed parameter to the top while keeping the other parameters below. This behavior leads to a progressive definition of all parameter values until an optimal configuration is reached. However, like any other stochastic optimization method it cannot be guaranteed, that the found solution is the absolute global optimum. Because of its nature of using random samples, it persists on a probabilistic behavior. This also entails that starting the algorithm multiple times will converge to different solutions. All algorithmic parameters have to be tuned in order to minimize these fluctuations.

The ratio between exploration and exploitation is quite fixed in the procedure outlined above as it is not intended to reassess different nodes of already fixed parameters when more knowledge is available. This behavior of an independent dynamically changing tendency to exploration or exploitation of the design space will be added in a future enhancement of this algorithm.

E. Algorithm Flowchart

The complete algorithm is shown in the flow chart depicted in Fig. 9. It is composed of several episodes that correspond to fixing the value of one parameter each. Thus the number of episodes is equal to the number of optimization parameters. The first episode is started after the initialization of all parameters, their respective boundaries and the optimization constraints.

A large number of random paths through the search tree is generated and stored. Each path is used in one event by selecting the parameter values according to the path, evaluating the performance and storing the calculated reward for each node that has been passed. After all random paths have been evaluated there is a decision node determining whether enough events have been successful in order to start processing the parameters and fixing a parameter value.

Another batch of random paths will be assessed as long as the number of successful evaluations is lower than the target quantity. A good value for this target quantity $N_{TargetEstimations}(N_{TE})$ is subject to the intended accuracy of the algorithm as well as the overall problem behavior. The subsequent estimation after N_{TE} is reached will create a weighted average of the reward seen by each node which implies that each node of every parameter has to be passed several times in order to smooth outliers. N_{TE} is defined by the following formula by taking the maximum number of nodes in a single optimization parameter P_j into account:

$$N_{TE} = max(nNodes_{P_j}) \times EF \tag{3}$$

EF is an evaluation factor that will be assessed in the following chapter by setting it to different multiples of the number of optimization Parameters. This effectively dictates that even for the parameter with the most discrete nodes, every node is visited at least EF times prior to the parameter assessment procedure.

This process of event evaluation can be easily distributed to multiple cores for parallel computing. A large number of events has to be computed for problems that produce a large amount of unsuccessful evaluations due to narrow constraints or for problems with lots of parameters which makes the overall algorithm very efficient when distributed over a large number of cores.

Next, the parameters will be rated by generating the Value-Function V and enhancing the parameter resolution for the continuous parameters. This is used to determine the most promising and influential parameter which will be fixed to the most promising value thus reducing the search tree.

The next episode will start at the top of the flowchart as long as there are open parameters leading to a fast convergence of the algorithm to an optimal value.

III. Computational Results and Application

A. Tuning of algorithm settings

Several settings used in the algorithm have been tuned by first tests. The impact of these settings will be assessed in more detail. The most influential parameter with respect to convergence speed is N_{TE} because it dictates how many events have to be evaluated successfully prior to the assessment of the results. N_{TE} is dependent on the Evaluation Factor EF and the maximum number of nodes in a single parameter. Truly discrete parameters will mostly only consist of a very small number of different possibilities which is why the maximum number of nodes is usually equal to the nodal count of quasi-continuous states which is typically set to 10-15.

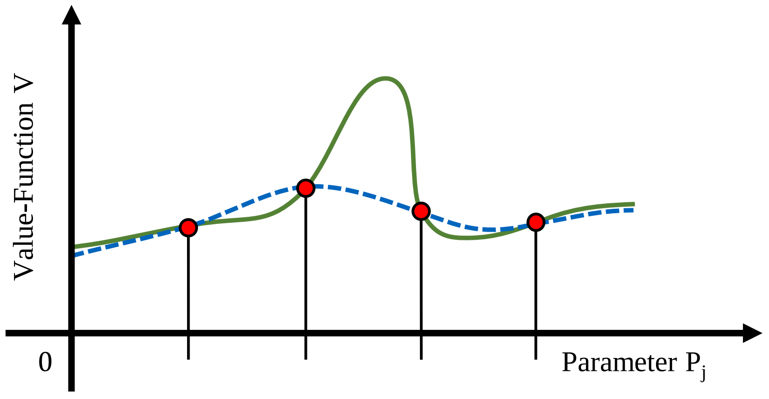


Fig. 10 Depiction of too low discretization count.

The influence of different nodal spacings can be seen in the middle sub-plot of Fig. 6b and Fig. 7. The first graphic was created using 60 nodes for all three quasi-continuous parameters while the latter was created using only 15 nodes. In direct comparison it can be clearly seen that many nodes do not necessarily increase the accuracy as the noise generated by many nodes will be filtered either way when applying a regression algorithm on the V-Value. However, reducing the number of quasi-continuous nodes below approx. 5 will prevent local peaks in the parameter space being detected, which is illustrated in Fig. 10. The peak that is present in the real reward function depicted by the green line is very narrow and only has a very local influence on the overall response function. It will not be detected by the sparsely distributed nodes indicated by the red dots. Furthermore it will also not be recognized by the regression drawn as a blue dashed line. The parameter assessment described above would detect a peak close to the second discretization point on the left and thus miss the true maximum. This also demonstrates an inherent problem to global optimization methods that cannot ensure to predict a global optimum. The result will be close to the global maximum when the design space is sufficiently smooth. Very narrow local maxima will however not certainly be found.

EF can be chosen quite freely compared to the nodal spacing when setting N_{TE} depending on the complexity of the problem. As more optimization parameters increase the size of the design space in an exponential way, it is advisable to bind EF to the number of optimization parameters $nOptParam$. N_{TE} is therefore redefined according to Eq. (4)

$$N_{TE} = max(nNodes_{P_j}) \times nOptParam \times EF \tag{4}$$

A study has been conducted were EF ranges from 0.2 to 15 with the sample points defined at:

$$EF = [0.25,\ 0.5,\ 1,\ 2,\ 3,\ 5,\ 10,\ 15] \tag{5}$$

As expected, the computational cost which is defined as the number of target function evaluations of the algorithm that is shown in Fig. 11 is increasing linearly with EF. Fig. 11 shows this number over the different EF defined in Eq. (5). Each line represents one optimization run for each EF.

The first few episodes are dominated by a large share of unsuccessful evaluations due to broken constraints. Fig. 12 shows the mean Validity Index of EF over all episodes. The first 6 episodes are characterized by a sharp increase of this index which is caused by the algorithm fixing parameters that produce lots of constraint violations on parts of their design space. The lines diverge after the first 6 episodes, however they all approach a validity index of 1 for the last few episodes. Looking at the number of function evaluations per episode in Fig. 14 shows the same phenomenon. The first episode exhibits the most uncertainties while the second and third episodes show an exponential decrease that can clearly be observed up to the 6th episode where the steepness of the Validity Index development of Fig. 12 is decreasing. This is true for all EF as is indicated by the bars at each episode number.

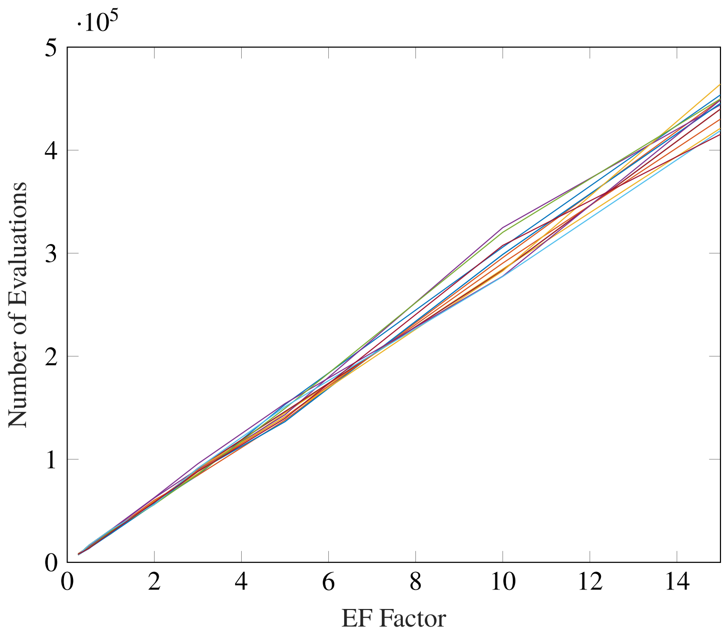


Fig. 11 Computational cost

A more detailed look on the later episodes in Fig. 15 shows the number of evaluations approach N_{TE} and is not changing greatly over the further episodes. To conclude, the convergence behavior does not differ much with different EF but the time taken will vary either way as the total number of evaluations is increasing linearly with EF.

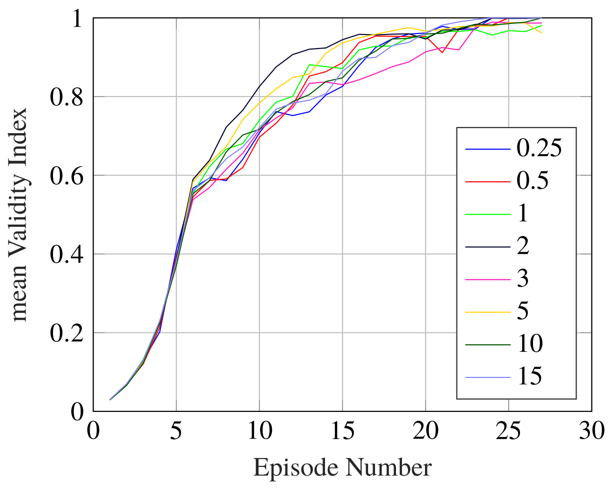


Fig. 12 Event validity.

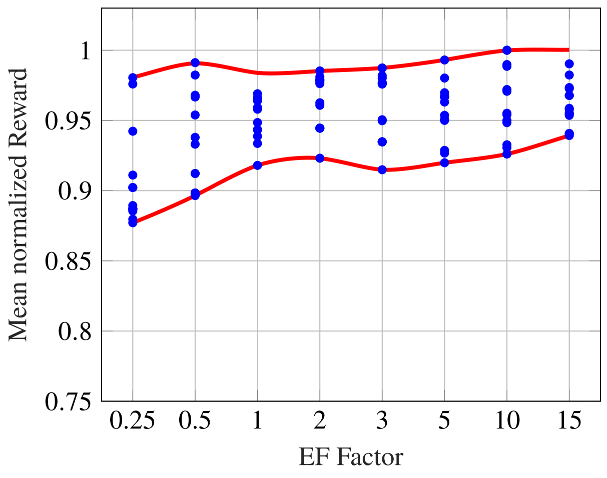


Fig. 13 Variance in predicted values.

However, the maximum predicted reward value is also varying which is affecting the robustness of the algorithm. Fig. 13 shows the deviation of the result for different EF-Values over all algorithm evaluations. A smaller EF will on average produce a marginally smaller reward while at the same time exhibiting a greater variance in the results. On the other end of the scale, the variance experienced by the 14 samples that have been calculated for this study is also higher than for $EF \approx 1$ or 2.

B. Convergence Rate and Scaling Behavior

Tying N_{TE} to $nOptParam$ has implications on the overall convergence rate of the optimization procedure. In general, according to the flow chart in Fig. 9, it will undergo as many episodes as there are optimization parameters.

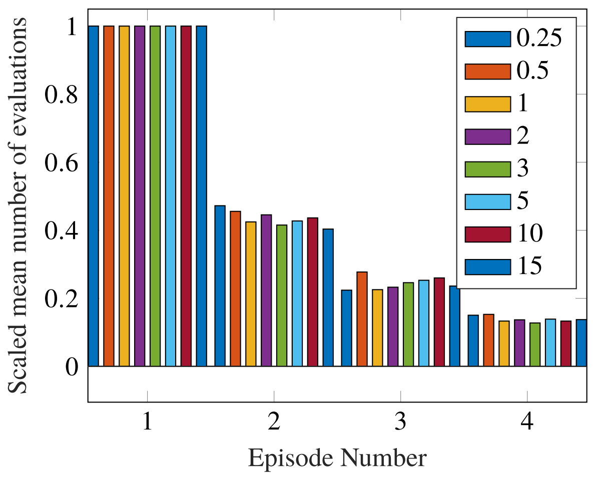


Fig. 14 Number of evaluations in first episodes

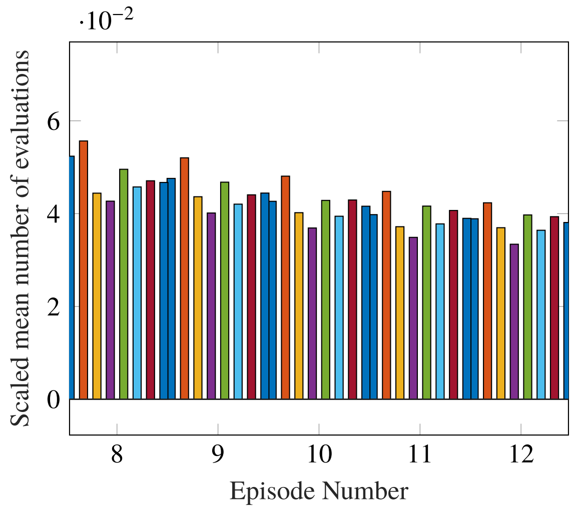


Fig. 15 Number of evaluations in later episodes

The length of one episode is influenced by N_{TE} as well as the overall validity index of the reward calculation. If many actions will violate constraints leading to a large share of unsuccessful actions, the total number of events per episode will be large. This can be seen in Fig. 14 by the bars of the first few episodes being much higher than the later ones. The validity index drawn over all episodes in Fig. 16 shows an explanation for this behavior. The validity index of the first few episodes is very small as this optimization problem is rather ill posed. However, the fixation of the first few parameters will weed out many solutions that violate constraints and thus results in a sharp increase within few episodes. The total number of target function evaluations shown in Fig. 14 will then approach N_{TE} quickly which is indicated by the validity index approaching 1. The parameters with less influence on the result and with no means to result in broken constraints will be fixed during the last episodes. The large additional unsuccessful evaluations in the beginning are independent of the number of parameters but are rather influenced by the quality of the design space and the characteristics of the constraints. Many constraints like the violation of a maximum blade tip speed can be considered even before the actual prediction of the propeller performance. This will lead to the unsuccessful actions taking much less time than a successful evaluation. An ill-defined design space with constraints that can only be evaluated after calculating the propeller performance like a minimum thrust requirement will greatly increase the overall calculation time and reduce the algorithm's effectiveness.

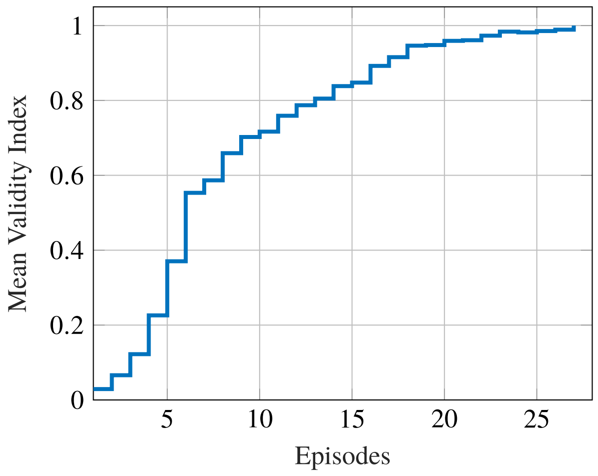


Fig. 16 Validity index development.

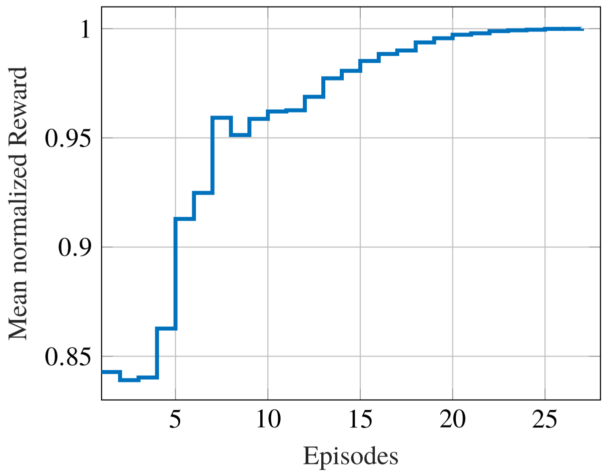


Fig. 17 Normalized mean reward per episode.

Another factor for the overall convergence time is the duration of one episode, which is scaling in a linear manner

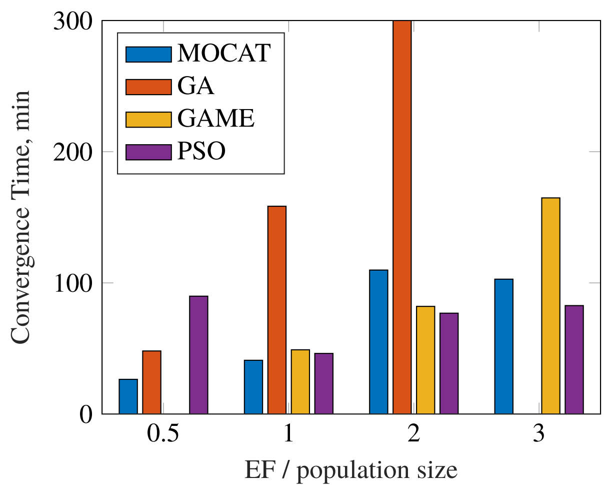


Fig. 18 Computational cost of the algorithms.

Table 4 Mean normalized reward value.

	Population Size			
	0.5	**1**	**2**	**3**
MOCAT	0.739	0.850	0.886	0.854
GA	0.836	0.748	1	-
GAME	-	0.905	0.859	0.951
PSO	0.860	0.865	0.869	0.866

with respect to $nOptParam$ according to Eq. (4). This leads to MOCAT currently scaling by $O(N \times N) = O(N^2)$ operations with N being the number of optimization parameters. Further work on the fine tuning of Eq. (4) and the parameter assessment process might further reduce this scaling effect. It is also expected, that Eq. (4) can be modified to no explicit dependence on $nOptParam$ as this number grows. This correlation has to be investigated in more detail.

Fig. 17 shows the mean reward of all successful events that was learned during the episodes normalized to the final reward value of the converged solution. This shows that the first few episodes do not only increase the validity index but also increase the predicted reward greatly. This figure is very familiar to the convergence behavior of other optimization algorithms. However, MOCAT will not stop its convergence when the overall rate of change in the reward tends to zero but will rather stop after it gained enough confidence to fix all parameters successively. Fig. 17 indicates that this a productive optimization approach.

C. Validation using Other Optimization Algorithms

The optimization results are validated by comparing them to the results obtained when different common optimization procedures are applied to the same problem. Three algorithms are chosen for this benchmark test:

- GA - A basic genetic algorithm's implementation that is included in the Matlab Global Optimization Toolbox [6]
- GAME - An improved genetic algorithm that has been developed by [1]
- PSO - A Particle Swarm optimization algorithm using an implementation that is also included in the Matlab Global Optimization Toolbox [6]

All three algorithms belong to the class of evolutionary algorithms that use stochastic methods to find an optimal solution. The most prominent configurational parameter for each of these algorithms is the population or swarm size which corresponds to the number of workers used in one generation or step. Its influence on the computational cost is similar to the influence of EF on MOCAT.

Each of these algorithms has been tested on the propeller design problem described above with different population sizes. Each test has been conducted four times in order to compensate for stochastic fluctuations. Fig. 18 shows a comparison of the computational cost of using these algorithms. Cost is growing as the population size is increased. Basic genetic algorithms like GA are known to have bad scaling behavior for many design parameters [7]. PSO also experience these scaling problems according to [8] although it is not that evident here. GAME algorithm is a better genetic algorithm and shows much better performance for problems with many parameters which can be observed in Fig. 18 by comparing the bars for GAME and GA. The computational cost of MOCAT is similar to that of GAME algorithm for all EF.

Looking at the normalized reward depicted in Table 4 does not show a true best algorithm for this sample study. PSO delivers steady results without much variation over the population size. MOCAT and GAME deliver comparable propeller efficiencies with GAME showing some fluctuations over the population size. MOCAT shows rather steady results for $EF > 0.5$ which can also be seen in Fig. 13.

The reward calculated with GA is rather small except for $EF = 2$ which is quite high but comes with a very high

Table 5 Aircraft optimization parameters for ADEBO.

Design Variable	Design Space
Wing Taper Ratio	0.1 – 0.9
Wing Sweep Angle	30 – 50 [*deg*]
Wing Aspect Ratio	6 – 12
Horizontal Tail Taper Ratio	0.2 – 0.6
Horizontal Tail Sweep Angle	20 – 45 [*deg*]
Horizontal Tail Aspect Ratio	3 – 5
Number of Engines	2, 4

Table 6 ADEBO optimization results.

Algorithm	mean MTOW	mean Computational cost
GAME	1	1
MOCAT	1.075	1.14

computational cost.

D. Application to a Holistic Aircraft Design Optimization

The propeller optimization problem discussed so far is characterized by some distinct properties that are of relevance to the algorithm convergence behavior.

- There are many (27) parameters which poses challenges for common optimization algorithms as seen in the previous chapter.
- The problem is quite ill-conditioned which is evident by the low validity index that is present, especially for the first few episodes as shown in Fig. 16.
- One evaluation of the target function described by the propeller efficiency takes only fractions of a second. This makes it possible to evaluate many possible combinations and enables a fast convergence of the algorithm despite the characteristics described above.

In order to assess the applicability of the optimization procedure to other design tasks, it is tested in designing an optimized transport aircraft using the interactive modular aircraft design framework "ADEBO" that was developed in [9] and which is based on an object-oriented knowledge based data model ADDAM (Aircraft Design DAta Model) [10].

The design parameters selected for this comparison are taken from a comparative study of several optimization algorithms on this task conducted in [11] are given in Table 5. The optimization goal was to minimize the Maximum Takeoff Weight $MTOW$ while not violating a size constraint by a too large wingspan and while fulfilling a given design mission. This design task exhibits quite contrary characteristics by having only 7 parameters, having less interdependencies in its design parameters and taking much more time for one target function evaluation. One evaluation of the ADEBO framework will take between 10 and 20 minutes on one core of a standard workstation which is at least 2000 times more than the propeller design evaluation. An optimization algorithm requiring many evaluations of its target function poses a problem to this task.

The results of comparing the MOCAT optimizations with the GAME algorithm which has been shown to be very well suited for the aircraft design problem in [11] are shown in Table 6. The numbers are normalized for the values achieved by the game algorithm. The results of the GAME optimization are taken from a study conducted in [11]. EF was chosen for the computational cost being roughly comparable to those of the GAME algorithm. The MOCAT design converged to a MTOW being 7.5% more heavy than the result from the GAME optimization. This difference in the result could be reduced by further increasing EF which would lead to a higher computational cost.

These sample tests do not yield generalized results on the performance of this algorithm compared to the other algorithms tested on this problem. However it is clear that problems with less parameters and cost intensive target functions can also be optimized. Nevertheless it can be concluded that this reinforcement learning based algorithm is best suited for problems with many optimization parameters that allow a fast evaluation of the target function due to the scaling behavior described above.

IV. Conclusion

The MOCAT algorithm demonstrates the successful application of reinforcement learning procedures on a physics-based design task. It is especially suited to optimize design parameters when the design space is large and the target function can be evaluated rapidly. It enables the seamless integration of multiple design goals and constraints by a freely defined target function that is calculating the reward. The results show that the current algorithmic architecture can outperform commonly used optimization procedures by converging in a rather short computational time to a good solution while the performance in the current implementation is comparable to other more sophisticated optimization procedures.

Further work is required for more detailed benchmarking studies on a broader range of optimization problems to further determine the algorithm's strengths and weaknesses. The current architecture allows for future improvements to the algorithm that have the potential to enhance it's optimization capability such as a finer tuning of the parameter assessment process and the incorporation of a more dynamical transition procedure between the exploration and the exploitation of the design space.

References

[1] Langer, H., "Extended Evolutionary Algorithms for Multiobjective and Discrete Design Optimization of Structures," Dissertation, Technical University of Munich, Garching, 2005.

[2] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An introduction*, [nachdr.] ed., A Bradford book, MIT Press, Cambridge, Mass., 2010.

[3] Alpaydın, E., *Introduction to machine learning*, 3rd ed., Adaptive computation and machine learning, MIT Press, Cambridge, Mass., 2014.

[4] Thiele, M., Obster, M., and Hornung, M., "Aerodynamic Modeling of Coaxial Counter-Rotating UAV Propellers," *8th Biennial Autonomous VTOL Technical Meeting & 6th Annual Electric VTOL Symposium*, edited by Vertical Flight Society, 2019.

[5] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D., "Mastering the game of Go without human knowledge," *Nature*, Vol. 550, No. 7676, 2017, pp. 354–359. doi:10.1038/nature24270.

[6] The Mathworks Inc., "MATLAB - Global Optimization Toolbox," , 2019. URL `https://de.mathworks.com/products/global-optimization.html`.

[7] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

[8] Piccand, S., O'Neill, M., and Walker, J., "On the scalability of particle swarm optimisation," *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, IEEE, 2008, pp. 2505–2512. doi: 10.1109/CEC.2008.4631134.

[9] Herbst, S., "Development of an Aircraft Design Environment Using an Object-Oriented Data Model in MATLAB," Dissertation, Technical University of Munich, Garching, 2017.

[10] Herbst, S., and Hornung, M., "ADDAM: An Object Oriented Data Model for an Aircraft Design Environment in MATLAB," *AIAA Aviation*, edited by AIAA, [publisher not identified], [Place of publication not identified], 2015. doi:10.2514/6.2015-3243.

[11] Kohestani, V., "Integration and Analysis of Different Optimization Methods for the Aircraft Design Box ADEBO," Semesters-thesis, Technical University of Munich, Garching, 2019.