Machine Learning for Flapping Wing Flight Control

Goedhart, Menno; van Kampen, Erik-Jan; Armanini, Sophie; de Visser, Coen; Chu, Qiping

DOI
10.2514/6.2018-2135

Publication date
2018

Document Version
Accepted author manuscript

Published in
Proceedings of the 2018 AIAA Information Systems-AIAA Infotech @ Aerospace

Machine Learning for Flapping Wing Flight Control

Menno W. Goedhart*, Erik-Jan van Kampen†, Sophie F. Armanini‡,
Coen C. de Visser† and Qiping Chu§
Delft University of Technology, 2629 HS Delft, The Netherlands

Flight control of Flapping Wing Micro Air Vehicles is challenging, because of their complex dynamics and variability due to manufacturing inconsistencies. Machine Learning algorithms can be used to tackle these challenges. A Policy Gradient algorithm is used to tune the gains of a Proportional-Integral controller using Reinforcement Learning. A novel Classification Algorithm for Machine Learning control (CAML) is presented, which uses model identification and a neural network classifier to select from several predefined gain sets. The algorithms show comparable performance when considering variability only, but the Policy Gradient algorithm is more robust to noise, disturbances, nonlinearities and flapping motion. CAML seems to be promising for problems where no single gain set is available to stabilize the entire set of variable systems.

Nomenclature

c	Vertical speed, m/s
F	Force, N
f	Flapping frequency, Hz
g	Gravitational acceleration, m/s^2
I	Moment of inertia, kg m^2
J	Cumulative reward, -
m	Mass, kg
q	Pitch rate, rad/s
S	Score, -
U	Reward, -
u	Control input, -
$\mathfrak{u}$	Out-of-plane velocity, m/s
V	Horizontal speed, m/s
w	Axial velocity, m/s
x	State
α	Angle of attack, rad
δ	Deflection, rad
θ	Pitch angle, rad
ϕ	Nonlinearity, -

Subscript

d	Disturbance
e	Elevator
n	Noise

*MSc Student, Control & Simulation Section, Faculty of Aerospace Engineering, Kluyverweg 1, Delft, The Netherlands.
†Assistant Professor, Control & Simulation Section, Faculty of Aerospace Engineering, Kluyverweg 1, Delft, The Netherlands. AIAA Member.
‡PhD Student, Control & Simulation Section, Faculty of Aerospace Engineering, Kluyverweg 1, Delft, The Netherlands. AIAA Student Member.
§Associate Professor, Control & Simulation Section, Faculty of Aerospace Engineering, Kluyverweg 1, Delft, The Netherlands. AIAA Member.

I. Introduction

The DelFly[1] is a Micro Air Vehicle (MAV) that generates lift by flapping its wings. It has been capable of flight since 2005. Several authors have been able to develop automatic flight control systems for the DelFly by means of PID control.[1–4] However, these controllers are all limited to specific flight conditions. Only for flight in a wind tunnel, more advanced control concepts have been applied.[5]

Due to the small size of the DelFly, manufacturing is very complicated, which causes variations between individual vehicles.[6] In practice, it is found that the gains of the controllers need to be hand-tuned for each individual DelFly, which is time-consuming.[6]

The application of more advanced controllers has been limited by the unavailability of accurate aerodynamic models: the first linear model was published only in 2013.[7] Research efforts have led to accurate models for specific flight conditions,[8] but the identification of a global model until recently has been challenging.[9] Adaptive or robust controllers are required to deal with the variability and nonlinearity of the DelFly.

Reinforcement Learning (RL) control tries to mimic the learning methods that humans and animals use.[10] Reinforcement Learning is a Machine Learning (ML) technique where the intelligent system learns by trial and error. Historically, RL has been used to study the learning behavior of animals.[11] From a control engineer's perspective, it provides a trial and error approach to optimal control, where the performance of a poor controller is improved continuously. The strength of Reinforcement Learning methods is that they can often deal with an inaccurate model of the controlled system. This makes RL suitable for problems where accurate models are not available.[6] Reinforcement Learning has been applied to flapping wings to maximize lift,[12–15] but has never been used for flight control of Flapping Wing MAVs.

This paper is a step towards a more intelligent controller for the DelFly, using Machine Learning. RL control theory is employed to develop a Proportional-Integral (PI) controller that automatically tunes its gains after gathering flight experience.[6,16] Additionally, the Classification Algorithm for Machine Learning control (CAML) is proposed, a novel classification controller where a Neural Network (NN) is used to select the gains from a predefined set. This is compared to the performance of a conventional PI controller with fixed gains. Simulations are performed to assess the performance of these algorithms.

The following structure is used in this paper. Section II describes the problem of controlling the DelFly, which is used to compare the algorithms. The Policy Gradient (PG) algorithm is described in section III, and CAML in section IV. The results of the comparison are presented in section V. Section VI concludes this work. The appendix describes how the nonlinearity measure used in this research is defined.

II. Problem Description

This research is both a step towards a new controller for the DelFly, and a comparison of Machine Learning control algorithms. For both goals, it is essential to understand the control problem, which is described in this section. The system to be controlled, on which the models used in this research are based, is shown in figure 1.

Several challenges were identified for the control of the DelFly. First of all, the *variability* between different individual vehicles calls for adaptive or robust control methods. Second, the dynamics of the DelFly are *nonlinear*. Third, the *flapping* motion of the wings influences control. Finally, *noise* and *disturbances* affect the possible controller performance.

This research focuses on automatic control: the vision-based control architecture is not used. As an intermediate step[a], a simulation of flight with feedback from an optical tracking system is performed. This research only considers control of the elevator. The throttle and rudder are not used in the simulations, because accurate models including these actuators are not available.

II.A. Dynamic Models of the DelFly

The dynamic models of the DelFly are the tools available to solve the challenges. Currently, these are mostly *local models*, which are valid in only one flight condition. The model structure[8] considered in this paper splits the dynamics into a time-averaged part, which is a Linear Time-Invariant (LTI) model, and a time-varying

[a]The DelFly is equipped with an Inertial Measurement Unit, and the optical tracking system provides drift-free state measurements. Such measurements could be provided outside the laboratory by the on-board camera or satellite navigation, but this is more complex.

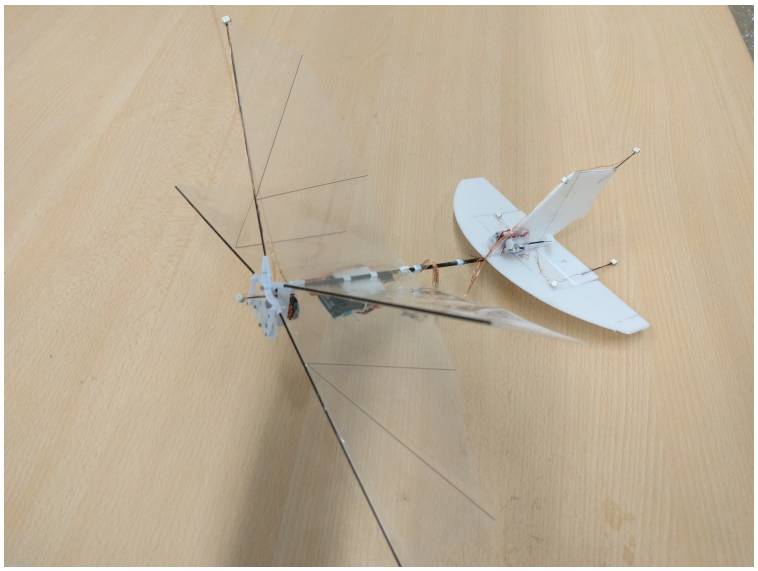


Figure 1: Variant of the DelFly II, on which the simulations in this research are based.

part, which is a Fourier series approximating the forces due to the repetitive flapping motion. These parts are used to match low-pass and high-pass filtered versions of flight data, respectively. The structure of the time-averaged model part is shown in Eq. 1.[17] The bias parameters b_1, b_2, and b_3 are not part of the actual system dynamics, but are only used for model estimation.

$$\begin{bmatrix} \Delta\dot{q} \\ \Delta\dot{u} \\ \Delta\dot{w} \\ \Delta\dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{M_q}{I_{yy}} & \frac{M_u}{I_{yy}} & \frac{M_w}{I_{yy}} & 0 \\ \frac{X_q}{m} - w_0 & \frac{X_u}{m} & \frac{X_w}{m} & g\cos\theta_0 \\ \frac{Z_q}{m} + u_0 & \frac{Z_u}{m} & \frac{Z_w}{m} & g\sin\theta_0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta q \\ \Delta u \\ \Delta w \\ \Delta\theta \end{bmatrix} + \begin{bmatrix} \frac{M_{\delta_e}}{I_{yy}} & b_1 \\ \frac{X_{\delta_e}}{m} & b_2 \\ \frac{Z_{\delta_e}}{m} & b_3 \\ 0 & b_4 \end{bmatrix} \begin{bmatrix} \Delta\delta_e \\ 1 \end{bmatrix} \tag{1}$$

There exist 46 of such models[17] for different flight conditions. Very recently, efforts have been made to merge these local models into one global model using a Linear Parameter-Varying approach,[9] where the airspeed and angle of attack are used for scheduling. This model is used in some simulations in this research.

For the time-varying part of the model,[8] a third order Fourier series is used, as shown in Eq. 2, where the base frequency equals the flapping frequency. Coefficients are available to predict the pitching moment, axial force, and out-of-plane longitudinal force. Only one set of coefficients is used in this research.

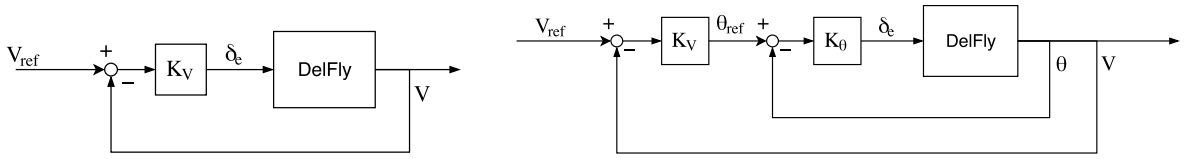
$$F_i(t) = \sum_{n=1}^{3} \left[a_n \sin(2\pi n f t) + b_n \cos(2\pi n f t) \right] \tag{2}$$

The full model is obtained by simply adding the partial models. The time-varying forces are scaled by their respective inertia terms and added to the accelerations in Eq. 1.

In this research, discrete-time models are used. A first order (Forward Euler) discretization is used, and the simulations are performed at 100 Hz. When flapping is introduced, however, the simulation time step is reduced to arrive at 500 Hz. The control algorithms are still run at 100 Hz to allow a fair comparison.

II.B. Control Objectives

The Reinforcement Learning algorithm can only learn if a single fixed task is repeated. The available models have only the elevator as an input, which is normally used for speed control[3] during hover and slow forward flight. Therefore, the natural control goal for this research is an airspeed controller. Altitude is not yet controlled, because accurate models accounting for throttle variations are not available. The task to be



(a) Single loop PI controller (PI)

(b) Cascaded PI controller (2PI)

Figure 2: Structures of the controllers used in this research.

performed by the airspeed controller is a 0.1 m/s increase in horizontal speed, when starting from a trimmed condition.

Proportional-Integral (PI) controllers are used for the underlying low-level controllers, because the integrating term facilitates trimming the DelFly. Two variants of the low-level controller are implemented, and shown in figure 2. The simplest, referred to as PI, takes as input the velocity tracking error, and processes this with a PI controller to calculate the required elevator deflection directly. The second variant uses two control loops. The velocity tracking error is fed to the outer PI controller, which specifies a pitch angle reference. The pitch angle tracking error is processed by the inner PI controller, which specifies the elevator deflection. This variant is referred to as cascaded PI, or 2PI.

Reinforcement Learning theory defines a *reward function*, which presents the controller with a reward on every time step. Within the framework of Markov Decision Processes, rewards can be based on the state and action taken, and may be probabilistic. For a velocity tracking controller, the most straightforward reward function is a quadratic punishment for deviations from the desired velocity. A punishment for control deflections may be added to limit the deflections.

If the goal of the controller is to maximize the cumulative reward, optimal controllers can be determined. Figure 3 shows the velocity V, pitch rate q, and elevator deflection δ_e for an example DelFly model with optimal controller for the reward function $U = -(V - V_{ref})^2$. It can be seen that the pitch motion is poorly damped. This seems to be due to the reward function, which only focuses on velocity deviations. If a penalty for q is added, better damping is obtained. The elevator deflection is sufficiently small, such that a punishment for control deflections is not required.

When flapping is introduced, there is a high-frequency vibration that cannot be fully suppressed by the controller. It was found that more favorable tracking is obtained when the controller is given knowledge of a filtered state. A fourth order Butterworth filter is used for this. Because this filter makes use of previous states, the control problem is not a Markov Decision Process. This may have implications when Reinforcement Learning methodology is applied to this system. Theoretical convergence analysis of the original RL algorithm in section III has only been performed[16] on Markov Decision Processes.

II.C. Simulation Methodology

A series of simulations is performed to assess the performance of the algorithms on the DelFly. The goal is to assess the abilities of the algorithms to address the five challenges. The first test considers only the variability: the algorithms are tested on the 46 local linear models that are available, without including flapping or disturbances in the simulation. Noise is fixed to the expected level, because zero noise would be unrealistic.

This tracking test is used as a baseline for a number of sensitivity analyses. The sensitivity to measurement noise is investigated by scaling the noise in steps. The L_2-norm of the vector of measurement uncertainties divided by Root-Mean-Square (RMS) values of the states is used as a noise measure. Next, the sensitivity to disturbances (i.e., process noise) is addressed, by superposing a (Gaussian) random elevator deflection at every time step. The standard deviation of this deflection is varied. During the disturbance test, noise is again fixed, in order to allow a fair comparison with the baseline.

A third sensitivity analysis considers the sensitivity to nonlinearities. An artificial nonlinearity, defined in the appendix, is introduced to fulfill this analysis.

The effect of the flapping motion is analyzed by starting from the variability test. The time-varying model part is superposed on the time-averaged part for each of the 46 models. The control algorithms need to be extended with the filter to deal with this.

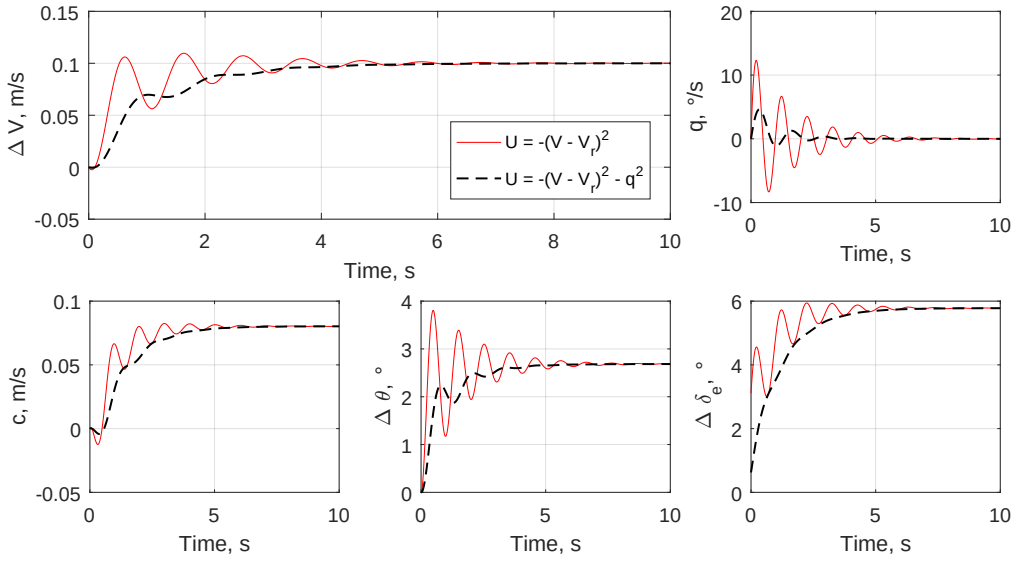


Figure 3: Effect of adding a pitch rate punishment on the optimal PI controller for an example system. The punishment improves damping, while still leading to effective tracking. Elevator deflections are sufficiently small in both cases.

A final analysis considers the recent global model of the DelFly. By introducing noise and flapping in the simulation, a test with this model can assess the performance of the control algorithms with respect to the challenges of noise, flapping and nonlinearity. Since there is only one such model available, assessing variability is not possible for the LPV model. Elevator disturbances are not introduced, since their expected magnitude is unknown, and the flapping motion is already a large disturbance.

The algorithms are assessed on their robustness to variability and tracking accuracy. The robustness to variability, i.e., safety, is defined by the fraction of controllers that is successful, i.e., obtains a cumulative reward higher than -100 (which is the reward obtained without any control) during every episode. Tracking accuracy is determined by the mean cumulative reward over ten episodes of all successful controllers. A combined score is obtained by giving all unsuccessful controllers a cumulative reward of -100, and averaging over all controllers.

III. Policy Gradient Algorithm

The first algorithm considered is a Reinforcement Learning algorithm of the actor-only category. This means that value functions, which are very common in RL, are not required. Instead, a parametrized actor is used to perform a repetitive task with a certain reward function. If the goal is to maximize cumulative reward, this is equivalent to a straightforward (stochastic) optimization problem, which can be solved using any optimization method. A gradient-based approach is used in this paper.

III.A. Using an inaccurate model

Determining the gradients is easier if a model is available during online control, because trial runs can be performed in simulation, while only using the best settings are used in the real system. This reduces the number of policies that are evaluated online, and thereby reduces the probability of using a dangerous policy. However, the model has to be accurate for this to work correctly.

In this paper, an algorithm[16] is used which uses an inaccurate model to suggest parameter changes, and uses the real system to obtain the cumulative reward. Algorithm 1[16] shows an implementation of this Policy

Algorithm 1 Actor-only approach using inaccurate models[16]

Input: T, with a locally optimal parameter setting ψ_+
Output: ψ

1: **repeat**
2: $\psi \leftarrow \psi_+$
3: Execute the policy with ψ in the real system, and record the state-action trajectory $x(0)$, $u(0)$, $x(1)$, $u(1)$, ..., $x(n)$, $u(n)$
4: **for** $t = 0, 1, 2, \ldots, n-1$ **do** {Construct a time-varying model $f_t(x, u)$ from $\hat{T}(x, u)$ and the trajectory}
5: $f_t(x, u) \leftarrow \hat{T}(x, u) + \left[x(t+1) - \hat{T}(x(t), u(t))\right]$
6: **end for**
7: Use a local policy improvement algorithm on $f_t(x, u)$ to find a policy improvement direction d
8: Determine the new parameters using $\psi_+ \leftarrow \psi + \alpha d$. Perform a line search in the real system to find α.
9: **until** ψ is not improved by the line search

Gradient (PG) method. The real system dynamics are represented by Eq. 3. This is approximated by an inaccurate model $\hat{T}$.

$$x(t+1) = T(x(t), u(t)) \tag{3}$$

This algorithm requires the construction of the time-varying model $f_t(x, u)$. $\hat{T}(x, u)$ provides a time-invariant representation of the system, which will (in general) not match the experienced trajectory exactly. By including the time-varying bias term $\left[x(t+1) - \hat{T}(x(t), u(t))\right]$ at every time step, $f_t(x, u)$ will match the experienced trajectory. The gradient obtained from $f_t(x, u)$ is then the gradient on the experienced trajectory.

For this algorithm, convergence to a local optimum in a Markov Decision Process can be proven if the local improvement algorithm is policy gradient.[16] However, this is only applicable if the true dynamics $T(x, u)$ are deterministic. Policy gradient algorithms similar to this one have been flight-proven using optical tracking systems,[6,18] which points towards suitability of the algorithm for the flight vehicle under investigation.

III.B. Variable learning rate

Performing a line search in the real system, as suggested in algorithm 1, results in a large number of policy evaluations. Furthermore, taking a step that is too large may result in unstable controller. A new algorithm with a virtual line search procedure is proposed in this paper. This is presented in algorithm 2.

This algorithm avoids time-consuming line searches by using a variable step size. An optimal step is size is found in the virtual system, and this step size is multiplied by a factor β for use in the real system. The learning rate β is decreased when the performance deteriorates. It has only one tunable parameter, which is the initial learning rate. In this research, the parameter step size is limited in order to increase safety.

When a flapping motion is present, the algorithm is adapted by consistently feeding the filtered state and reward to the controller. The unfiltered control deflection is used, because the deflection is based on the filtered state. The internal representation is left unchanged, i.e., the controller does not have the information that the state is filtered.

III.C. Controller behavior

An example of the behavior of the PG controller is shown in figure 4. Because of the actor-only approach used, the controller can only improve its gains once per episode. Nevertheless, it quickly reaches its optimal gains: the gains hardly change after four episodes. This controller is initialized with $\beta = 0.8$, and therefore gets close to the optimum after a single episode. The gains are fine-tuned during the later episodes.

The plots of airspeed and pitch rate reveal that the rise time decreases slightly, at the expense of a lower damping. When looking at the elevator deflection, it is found that the deflection signal is slow. The controller is not actively counteracting the oscillations, but relies on the natural damping of the vehicle. A different controller structure, such as the cascaded PI controller, is required for active damping.

Algorithm 2 Policy Gradient algorithm used in this research. This is a variation to algorithm 1.

Input: T, with an initial parameter setting ψ_+, and initial learning rate β
Output: ψ

Set $J_\psi \leftarrow -\infty$
repeat
  Execute the policy with ψ_+ in the real system, and record the state-action trajectory $x(0)$, $u(0)$, $x(1)$, $u(1)$, ..., $x(n)$, $u(n)$, with associated cumulative reward J_{ψ_+}.
  if ψ_+ improves over ψ **then**
    $\psi \leftarrow \psi_+$
    for $t = 0, 1, 2, \ldots, n-1$ **do** {Construct a time-varying model $f_t(x,u)$ from $\hat{T}(x,u)$ and the trajectory}
      $f_t(x,u) \leftarrow \hat{T}(x,u) + \left[x(t+1) - \hat{T}(x(t),u(t))\right]$
    end for
    Use a numerical algorithm on $f_t(x,u)$ to find the gradient $\nabla_\psi J$
    Determine new parameters using $\psi_- \leftarrow \psi + \alpha\nabla_\psi J$. Perform a line search in $f_t(x,u)$ to find the optimal α.
    Take a partial step towards ψ_- using $\psi_+ \leftarrow \psi + \beta\alpha\nabla_\psi J$.
  else
    Halve the learning rate $\beta \leftarrow 0.5\beta$.
    Take a smaller step towards ψ_- using $\psi_+ \leftarrow \psi + \beta\alpha\nabla_\psi J$.
  end if
until steps are sufficiently small.

IV. Classification Algorithm for Machine Learning Control

Because the Policy Gradient approach requires one full episode with the initial controller, it can never achieve a lower crash rate than a fixed PI controller. If greater tolerance to variability is required, the controller should be able to change its gains rapidly during an episode. Conventional adaptive Reinforcement Learning algorithms, such as Heuristic Dynamic Programming,[10] are expected to be unable to adapt fast enough to recover an initially unstable controller.

The task of controlling an unknown DelFly draws parallels with the 2008 and 2009 RL Competitions involving helicopter hovering. In this competition, the task was to control a helicopter with unknown dynamics.[19] The 2008 competition was won by a high-level controller selection approach of RL,[19] that requires all of N predefined controllers to be attempted on the real system, and selects the best one. This comes with the risk of attempting an unstable controller and crashing the system. Only safe controllers can be used, which are able to stabilize most DelFly systems. It is expected that the variability between different DelFly's makes it difficult to use the controller selection algorithm effectively.

A later algorithm for the RL Competitions uses a stochastic model in combination with Model Predictive Control.[20] This leads to accurate control and allows rapid changes in control parameters, but is computationally prohibitive.

This paper proposes a new approach, which combines the strengths of the controller selection approach and the stochastic Model Predictive Control approach. The basis of this approach is that an identified model, even if it has a large uncertainty, can be used to predict the effectiveness of the predefined controllers. Instead of performing time-consuming simulations to optimize a policy, the model is used directly to select one of the predefined controllers.

Figure 5 demonstrates the concept of this controller, named Classification Algorithm for Machine Learning control (CAML). A recursive model identification algorithm is used to identify the model online. Also, the parameter (co)variance of the model is identified, in order to obtain a measure of the uncertainty in the model.

The most straightforward approach with knowledge of the model and the parameter variance is to perform simulations with all available controllers for a wide range of models, whose distribution is defined by the identified mean model and covariance. However, performing these computations online is considered computationally prohibitive. Instead, a Neural Network (NN) is trained on a dataset of simulations that were

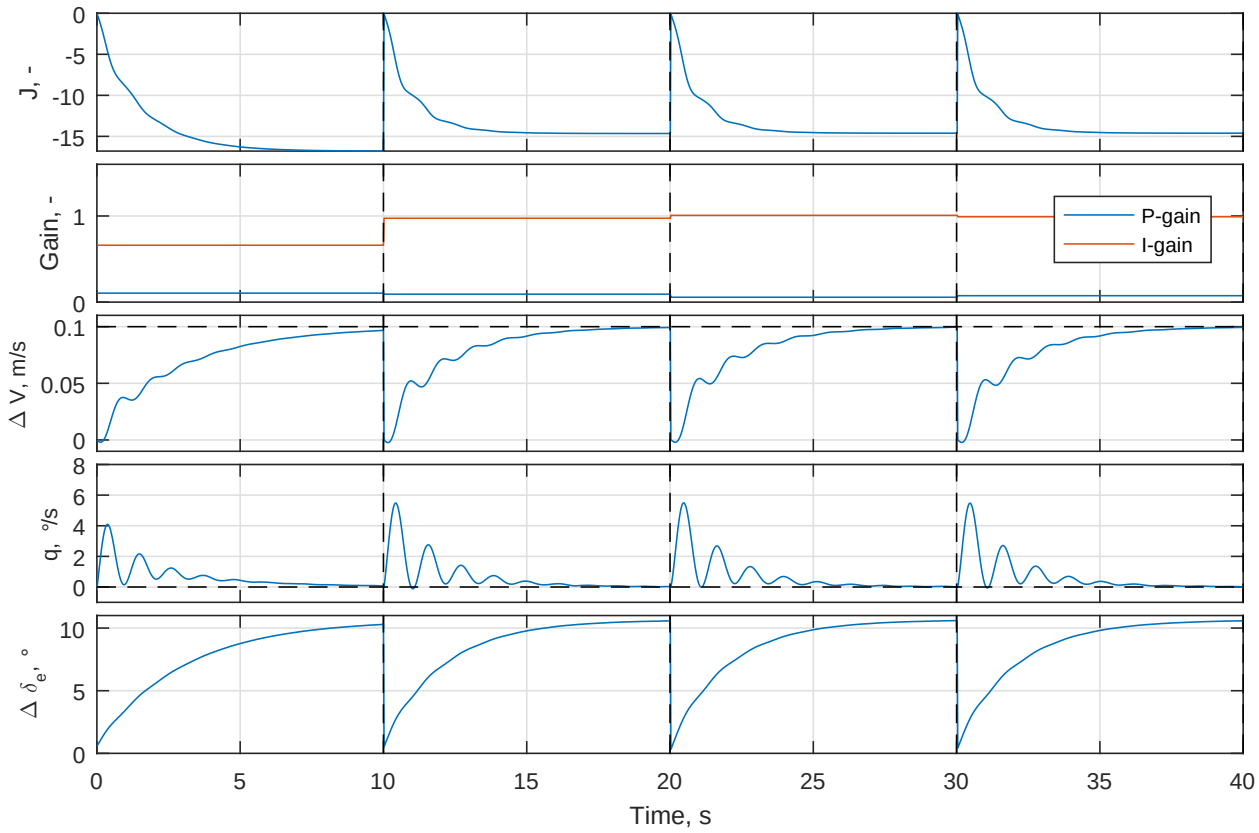


Figure 4: Example behavior of the Policy Gradient controller with PI structure when tracking a 0.1 m/s airspeed step, over four episodes. The increasing cumulative reward per episode (J) shows that the gradient-based learning procedure typically results in increasing performance over time, with changes once per episode.

performed offline. This NN takes the model and parameter variance as inputs[b], and outputs a prediction of the most suitable controller. This is a classification problem, where the combination of model and parameter variance must be allocated to one of N gain set categories. The softmax layer outputs N numbers, where each number represents the probability p_i that the input belongs to the corresponding gain set category. The gain set with the highest probability is selected on every time step.

CAML is intended to improve safety by allowing the gains to change rapidly. Upon a switch of gain set, the integral term is reset in such a way that the commanded elevator deflection remains constant. The CAML algorithm has a weak inherent protection against instability: if an unstable gain set is selected, the deviations from the trimmed state will become large. The inputs to the online model identification algorithm become large, so the identified model will change rapidly. If the new model is fed to the NN, it may select a different gain set. This mechanism does not result in guaranteed safety, but simulations show that it effectively improves safety.

Conventional gain scheduling methods predefine different linear controllers for a nonlinear system, where the state is used to switch between controllers. CAML can also be used for nonlinear systems, by identifying a linear approximation, on which the gain selection is based. However, CAML performs these computations during operation. This makes the algorithm suitable for unknown systems within the space of expected systems, whereas gain scheduling controllers are only suitable for the system they are designed for.

IV.A. Predefined gain set generation

The predefined gains used by CAML are tuned by examining the 46 known models. For every model, an optimal PI gain set is determined by means of an offline optimization routine (gradient descent with multiple

[b]The computational complexity of CAML is dependent on the number of states. This research only considers four-state models. A formal analysis is required to find a practical limit to the number of states.

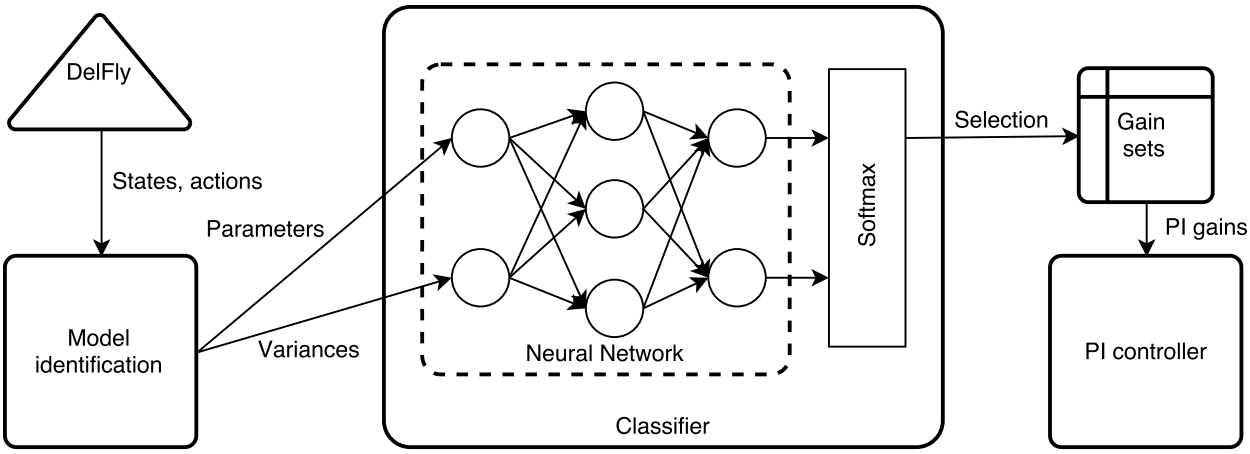
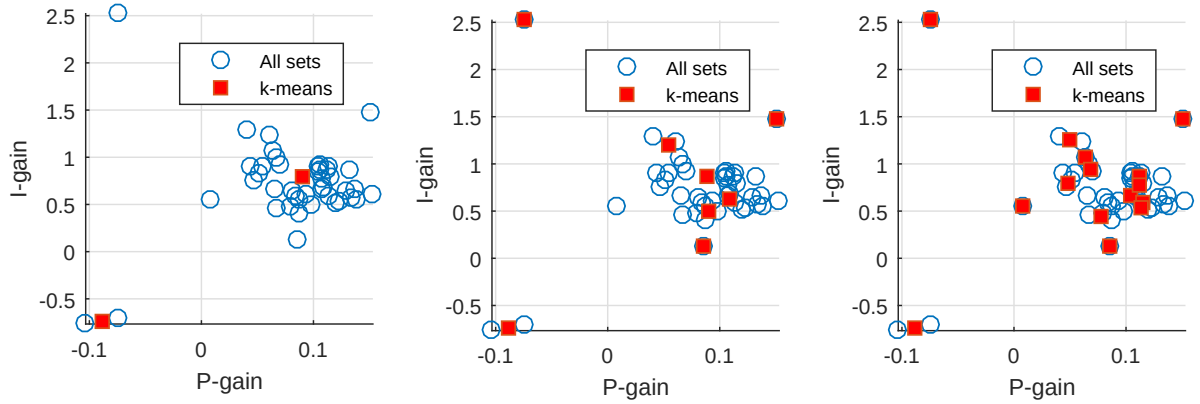


Figure 5: Principle of CAML. The parameters and variances of a model, identified online, are fed to a classification Neural Network, which selects the most appropriate gain set for the low-level controller.



(a) $k = 2$ (b) $k = 8$ (c) $k = 15$

Figure 6: Optimal PI controllers for the known models. The k-means clustering algorithm makes a smaller selection. It captures the outliers well, but cannot fully match the spread in the main cloud.

starts). These gains optimize the cumulative reward as defined in section II. The closed-loop stability of the resulting 46 PI controllers with their models are analyzed, and unstable gain sets are discarded.

The resulting gain sets[c] are shown in figure 6. For the selectable gain table, it is desired to have less than 46 options. A k-means clustering algorithm (k-means++[21]) is used to obtain a smaller representative set of gains. Figure 6 shows the resulting gain sets for the CAML controller for several k. The same process is followed for the cascaded PI controllers.

IV.B. Training the Neural Network

The classifier must be trained using a large number of examples. These examples are sets of model parameters, variances, and correct selections. The pairs of model parameters and variances are generated with the 46 available models. For every combination of models A and B, algorithm 3 is used. This yields series of aerodynamic coefficients starting at model B, with suitable initial variance[d], and slowly converging to model A with lower variance. The result is a set of realistic combinations of model parameters and variances.

For a deterministic set of training models, the training targets are determined by simulating all N_c gains

[c]There are two models with inverted control effectiveness, which results in negative optimal gains. The model with negative P-gain, but positive I-gain has normal control effectiveness, but is very different from the remaining models.

[d]The initial model variance is based on the variance in the aerodynamic coefficients of the 46 models.

Algorithm 3 Generation of pairs of model parameters and variances

Input: 3-2-1-1 response of DelFly A, with n data points $x_A(t), u_A(t)$.
Output: An array V of pairs of model parameters and variances.
1: Initialize a model M at the aerodynamic coefficients of DelFly B.
2: **for** $t = 0, 1, 2, \ldots, n-1$ **do**
3: Update M using RLS, with data point $x_A(t), u_A(t) \rightarrow x_A(t+1)$.
4: Save the parameters and variances of M to $W(t)$
5: **end for**
6: Take random samples of W, resulting in a smaller array V.

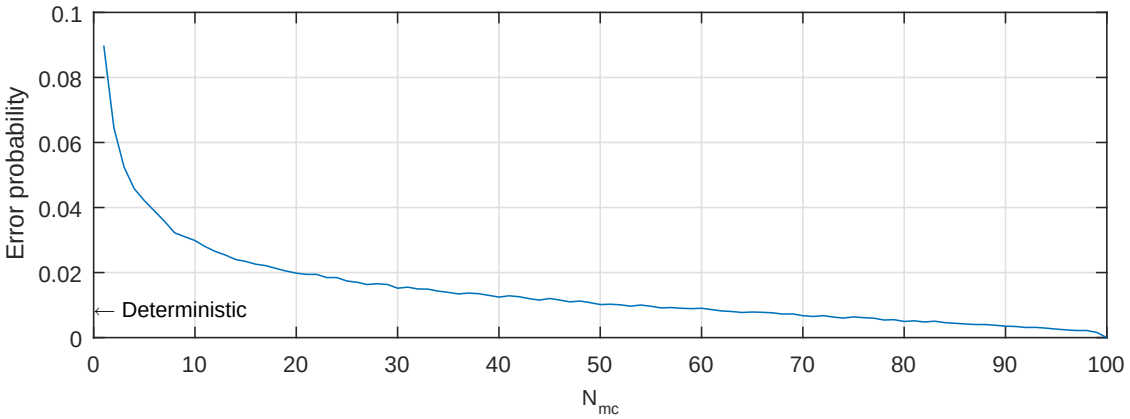


Figure 7: Error probability in the training data as a function of the number of random models. The best gain set when testing with $N_{mc} = 100$ is considered perfect. The low error rate of 0.009 if variance is disregarded indicates that a variance input is not necessary.

sets with all N_t models. The twelve important aerodynamic coefficients are the training inputs[e], and the training targets are the indices of the most suitable controllers. Training is performed using scaled conjugate gradient training with early stopping (80% training data, 20% validation data). Five independent NNs are trained, and the best performing one in terms of error rate on the complete dataset is selected.

An additional step is required to account for a nonzero variance. For each training model, N_{mc} new, random models are created, by adding Gaussian random values (scaled by the corresponding variance) to each of the aerodynamic coefficients. The optimal controller is determined by averaging the performance of each gain set over the N_{mc} random models[f]. In this case, the training input consists of twelve aerodynamic parameters and twelve variances.

The variance input to the NN ensures that the controller can select a safe gain set when the uncertainty is high. However, it adds twelve input dimensions to the NN and makes training more time-consuming, by a factor N_{mc}. The number of random models should be high enough, to avoid incorrect gain selections in the training data.

Figure 7 is used to analyze the appropriate value of N_{mc}. In this figure, it is assumed that selection is perfect with $N_{mc} = 100$[g], i.e., 100 random models related to each training model. This is repeated for 1000 training models, and the error probability in the training data with lower N_{mc} is analyzed. For comparison, it is found that the error probability is 0.009 if the variance is disregarded, and only the identified model is analyzed. In this particular analysis, at least 57 random models must be used to match this. The order of magnitude of the error rate achievable by the NNs on such datasets is 0.05. It is therefore decided that the variance input does not contribute effectively, and only NNs without variance input are considered further in this research.

[e]A complication is that the A-matrices of the models include both aerodynamics coefficients, which are inputs of the NN, and kinematic terms. The training simulations are performed using models with different aerodynamic coefficients, but equal kinematic terms, corresponding to a trimmed airspeed of approximately 0.8 m/s.

[f]If a gain set is unstable for a random model, a fixed penalty is given, in order to avoid excessive punishment of unstable models.

[g]This makes this analysis unreliable for N_{mc} close to 100.

The training models with variance are identified by letting RLS converge from one model to another. This is a rather limited procedure of random model generation, because the resulting models will always be close to the 46 known models. A more general procedure is to generate random training models, based on the means and variances of the aerodynamic coefficients of the 46 known models. For each coefficient, the mean and variance among the 46 models is determined. An independent Gaussian distribution for every parameter is assumed to generate new models. This is performed after repeatedly removing outlier models[h], which have an aerodynamic coefficient varying more than 3σ from the mean. The resulting training dataset covers a larger space of possible models, but NN training is more difficult, with resulting error rates in the order of 0.15. It is found that both methods of training model generation result in similar performance of the CAML controller when tested on the 46 known models.

IV.C. Model Identification

This method is highly dependent on the identification of a model of the dynamics of the DelFly. This section describes how this model is identified during operation. The model structure in Eq. 1 is used. For this application, it is more convenient to use the discrete-time model in Eq. 4, which is obtained by means of a first order (Forward Euler) discretization. Note that several matrix elements are set to zero, and some elements are dependent on the initial state only. Also, the last column of the input matrix does not contribute to the dynamics, but only to the trimmed condition. This results in twelve important aerodynamic parameters to be identified.

$$\begin{bmatrix} q(t+1) \\ \mathfrak{u}(t+1) \\ w(t+1) \\ \theta(t+1) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} - w_0\Delta t & a_{22} & a_{23} & g\cos\theta_0\Delta t \\ a_{31} + u_0\Delta t & a_{32} & a_{33} & g\sin\theta_0\Delta t \\ \Delta t & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} q(t) \\ \mathfrak{u}(t) \\ w(t) \\ \theta(t) \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ 0 & b_{42} \end{bmatrix} \begin{bmatrix} \delta_e(t) \\ 1 \end{bmatrix} \tag{4}$$

The online model identification problem is to determine the parameters of the matrices A and B from flight data, such that the model error $\hat{e}$ is minimized over time. This is done separately for every row of these matrices. As an example, Eq. 5 shows the (noise-free) identification problem for the pitch rate row.

$$q(t+1) = a_q(t)w_q + \hat{e}_q(t) = \begin{bmatrix} q(t) & u(t) & w(t) & \delta_e(t) & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_{11} & b_{12} \end{bmatrix}^T + \hat{e}_q(t) \tag{5}$$

An online model identification algorithm selects the weight vectors w such that the model errors $\hat{e}$ are minimized over time. The most straightforward method of online model identification is the Recursive Least Squares (RLS) algorithm.[22] This algorithm recursively solves for the parameters w in order to minimize measurement error e in problems of the form of Eq. 6. If $f(y, u)$ is linear, this problem is a special case of the auto-regressive moving-average model with exogenous input (ARMAX), which is shown in Eq. 7.

$$y(t) = \left[f(y(t-1), u(t-1))\right]^T w + e(t) \tag{6}$$

$$y(t) = \begin{bmatrix} -y(t-1) & \cdots - y(t-n_a) & u(t-1) & \cdots & u(t-n_b) & e(t-1) & \cdots & e(t-n_c) \end{bmatrix} w \tag{7}$$

The model identification problem can be seen as a vector form of the ARMAX problem, where the state vector $x(t)$ has to be predicted by the model. If there is a measurement noise vector $E(t)$, this problem can be written as in Eq. 8, where the measured state is denoted by $z(t) = x(t) + E(t)$.

$$\begin{aligned} x(t) &= Ax(t-1) + Bu(t-1) \\ z(t) &= Az(t-1) + Bu(t-1) - AE(t-1) + E(t) \end{aligned} \tag{8}$$

Because there is measurement noise at both t and $t-1$, the underlying assumptions of the RLS estimator are violated. If RLS is used anyways (implicitly assuming that $E(t-1) = 0$), the result will be inaccurate.

[h] This is required, because these outlier models cause unrealistically large variations among the newly generated random models. A disadvantage is that the trained classifier will be unable to perform accurate predictions for the outlier models.

The Recursive Maximum Likelihood algorithm RML1,[22,23] also called Extended Least Squares, is an option to solve this problem. This algorithm extends the regressor in RLS with the previous measurement error $e(t-1)$. Since this measurement error is not known, the prediction error $\hat{e}(t-1)$ is used instead. It is straightforward to extend this to the vector case.

Due to the structure of Eq. 8, the correct weights on $\hat{E}(t-1)$ and $z(t-1)$ are directly related, and can be constrained. Since $\hat{E} = z - \hat{x}$, this is equivalent to using the predicted previous state $\hat{x}(t-1)$ instead of $x(t-1)$ in the conventional RLS estimator. The resulting recursive algorithm is shown in Eqs. 9 to 13, where the regressor $a_i(x,u)$ is an affine vector function of the state and input (like $a_q(t)$ in Eq. 5).

$$r_i(t) = z_i(t) - w_i^T(t-1)a_i(\hat{x}(t-1), u(t-1)) \tag{9}$$

$$L_i(t) = \frac{P_i(t-1)a_i\left(\hat{x}(t-1), u(t-1)\right)}{a_i^T\left(\hat{x-1}(t), u(t-1)\right)P_i(t-1)a_i\left(\hat{x}(t-1), u(t-1)\right) + \lambda} \tag{10}$$

$$w_i(t) = w_i(t-1) + L_i(t)r_i(t) \tag{11}$$

$$P_i(t) = \frac{1}{\lambda}\left[P_i(t-1) - L_i(t)a_i\left(\hat{x}(t-1), u(t-1)\right)P_i(t-1)\right] \tag{12}$$

$$\hat{x}(t) = w_i^t(t)a_i\left(\hat{x}(t-1), u(t-1)\right) \tag{13}$$

Identification of the time-averaged model part was originally done by filtering with a fourth order Butterworth filter.[8] The non-causal zero-phase variant of this filter was used,[8] but this is not possible during online identification. This paper uses the causal variant of the Butterworth filter, which adds a phase shift, affecting model identification. Implementing the filter in the RML1 method is non-trivial, because the previous state $\hat{x}(t-1)$ is internal, while the current state measurement $z(t)$ is external. It is found that the conventional RLS algorithm is more resistant to this filtering effect than RML1. In this research, RML1 is used for all simulations without filtering, whereas RLS is used for simulations with filtering.

IV.D. Algorithm parameter selection

There are three hyperparameters governing the behavior of the classification controller: the number of selectable gain sets, the number of neurons in the hidden layer, and the number of training models. The optima of these parameters are clearly interrelated. Also, it needs to be considered that the gain sets and training models are based on the models that are also used for testing. This will give problems if the previously mentioned numbers are set too high.

Due to the time-consuming nature of the computations, a three-dimensional optimization on a fine grid is infeasible. Rather than optimizing on a coarser grid, it is opted to perform manual optimization, using engineering judgment. A sensitivity analysis on the selected near-optimal point is performed for every parameter individually. The baseline tracking test is used for tuning the parameters. All 46 models are used for optimization, because the number of models will also influence the optimal settings. Later, it is tested if splitting the data into a training set and a test set gives significantly different results.

IV.D.1. Number of gain sets

The number of gain sets in the controller should be high enough to cover the space of desirable gain combinations. A higher number should lead to higher possible performance. If the number becomes too large, however, the neural network will have problems making a correct selection, and the performance will decrease. While holding the numbers of neurons and training models fixed, figure 8 shows the performance for different numbers of gain sets.

The figure does not show the expected trend of performance with the number of gain sets. This is a first indication that CAML is not working optimally on this problem. Figure 6 provides an explanation for this behavior. If only two gain sets are used, CAML nearly always picks the gain in the middle of the main cloud in figure 6a. When then number of gain sets is increased, more outliers are captured, but the main cloud is not fully covered. It is found that fixed PI controllers with gains near the center of the cloud all lead to similar scores. Thus, increasing k will lead to more outliers in the gain sets, and more comparable gains. The latter consequence will tend to increase the score, but the first consequence will increase the risk of incorrect gain selection. It is expected that higher performance would be found if the gain sets are spread more evenly over the main cloud.

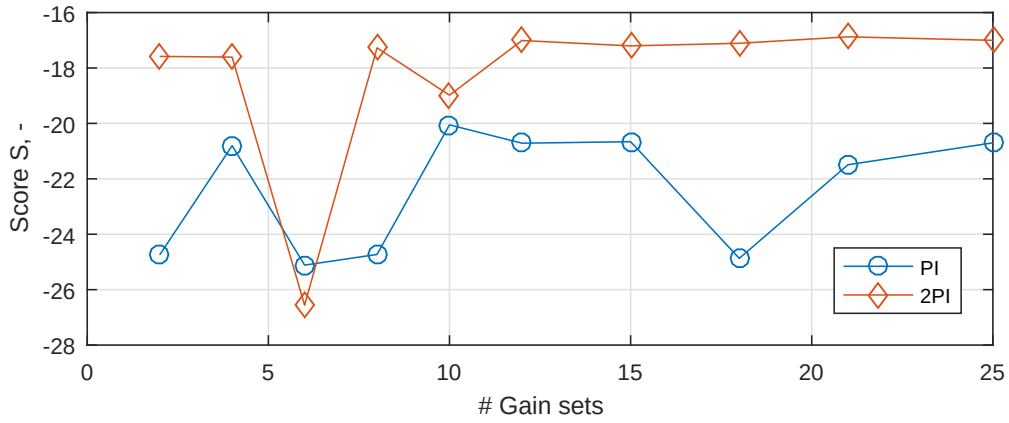


Figure 8: Tuning of the number of gain sets. Unexpected drops in performance are found for some numbers. Fifteen controllers is regarded near-optimal for both single and double loop PI controllers.

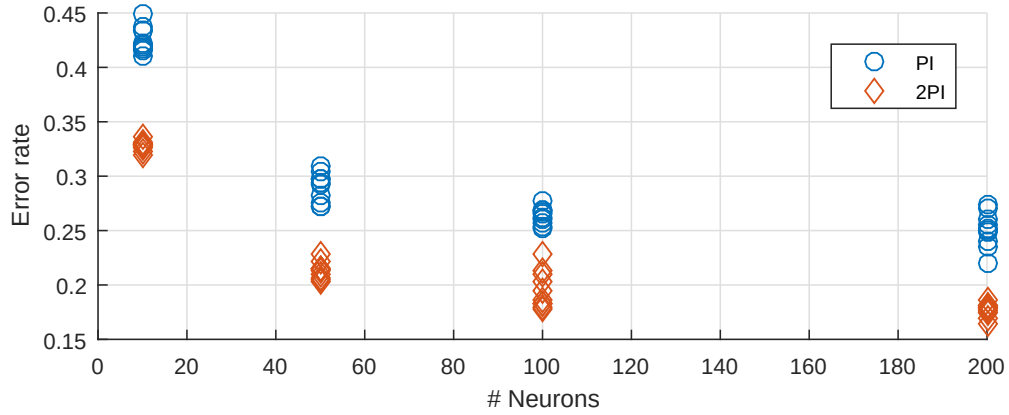


Figure 9: Tuning of the number of gain sets. Overfitting does not seem to be a problem. A number of 100 neurons is considered sufficient.

IV.D.2. Number of neurons

The number of neurons must be large enough to capture the complexity in the training data, but Neural Networks that are too large suffer from overfitting, and require longer computation times. The optimal number of neurons with respect to overfitting is analyzed by generating a new set of 10000 testing models. Neural networks of various sizes are trained on the training data, and tested on the testing data. This is repeated ten times, as shown in figure 9.

It can be seen that the error rate is steadily decreasing for the numbers of neurons considered. Since the test is performed on a separate dataset, this is an indication that overfitting is not yet problematic for these numbers of neurons. The error rates appear to stabilize at 100 neurons. Increasing the number of neurons further than 100 is not worth the computational time, so a number of 100 neurons is selected for both single and double loop controllers.

IV.D.3. Number of training models

Increasing the number of neurons will always result in increased performance, because more training data allows larger neural networks without overfitting, and thereby a more powerful classifier. However, the (offline) computational time limits the number of training models that is feasible in this research. Figure 10 analyzes the error rates on the 10000 testing models for different numbers of training models.

As expected, the performance keeps decreasing for larger numbers of training models. For high numbers, the error rate seems to stabilize. It is decided that 40000 models is sufficient for the purposes of this research.

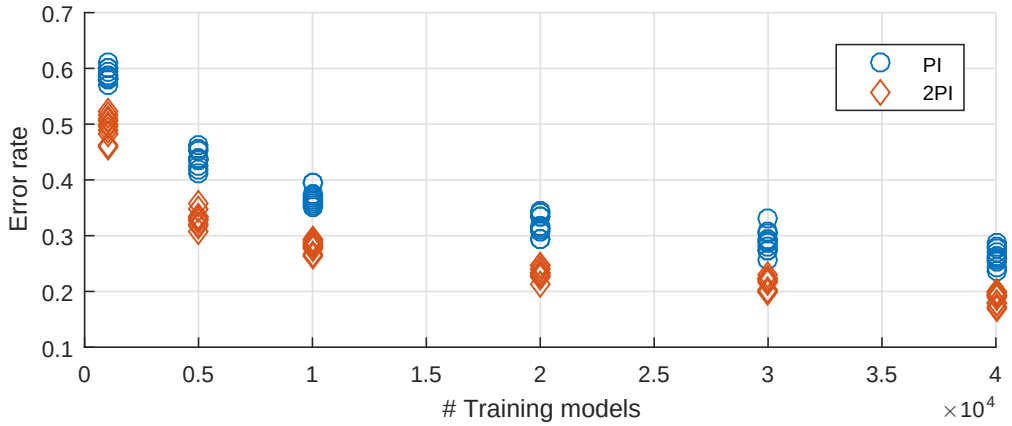


Figure 10: Tuning of the number of training models. It is decided that 40000 models is sufficient.

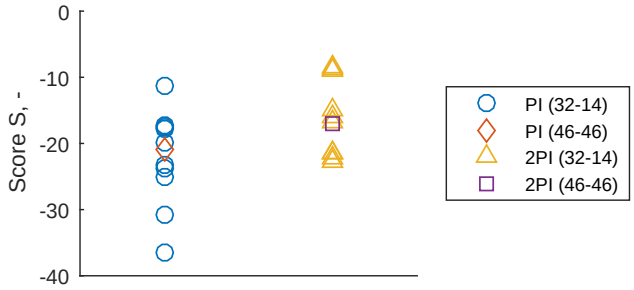


Figure 11: Variability results if 32 of the models are used for training, and 14 for testing, repeated ten times with random divisions. The controller trained and tested on the full dataset does not appear to be overly optimistic.

IV.D.4. Internal validation

It is common to split the data into training data and test data when testing tuned controllers, but this would limit the power of the classification controller. The allowable numbers of gain sets and neurons would have to be decreased. In this research, the tuning and testing are both performed on the full dataset. Afterwards, the data is split up several times, to investigate if the results with the full dataset are overly optimistic.

In this analysis, 70% (32) of the 46 models are used for tuning and training, and the remaining 14 are used for testing. The numbers of gain sets, neurons, and training models are left at their previously optimal values. The models in the training set is used to generate 40000 random models (disregarding the same outliers as before) on which new neural networks are trained, and to select appropriate gain sets. This is repeated ten times, with new divisions into training and testing data. The resulting scores are shown in figure 11.

It is found that the controller trained and tested on the same models is not unreasonably optimistic. There is no reason to assume that using the same models for training and testing is causing any problems.

IV.E. Controller behavior

Figure 12 shows an example of the behavior of the CAML controller with a PI structure. The example system is the same as in figure 4, and the figures can be compared directly. During the first seconds, the model, which is at the input of the NN, is uncertain and changes rapidly. This causes fast variations in the outputs of the NN, which are the softmax probabilities p_i for each gain set, which is reflected by the varying gains. The number of gain switches decreases during later episodes, due to the slower changes in the model. Gain switches are concentrated in the first seconds of every episode, because the excitation of the system is largest in these periods.

A disadvantage of the CAML algorithm is that the performance does not necessarily improve over time. The identified model improves, but the NN classifier has a nonzero error rate, and can therefore select an

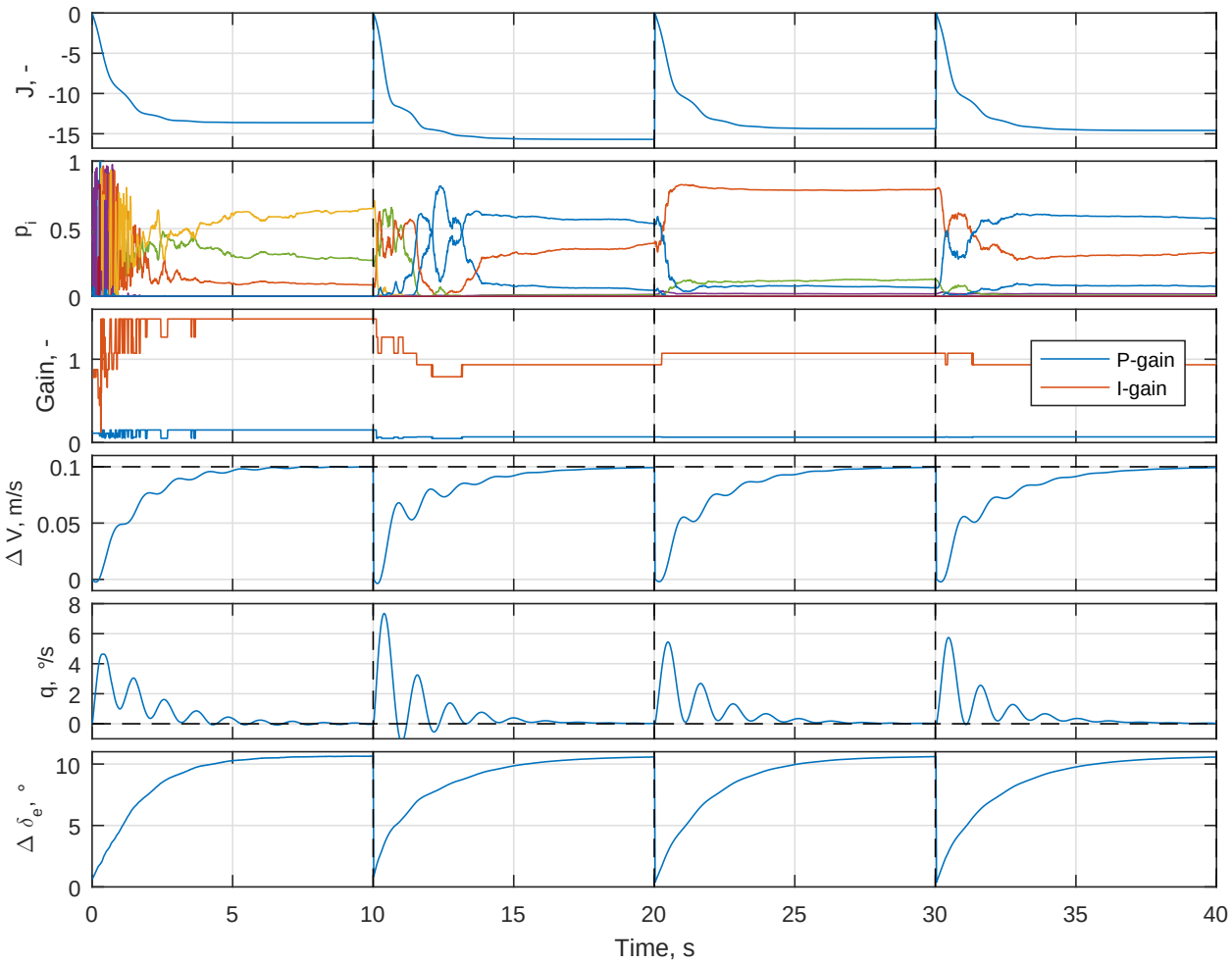


Figure 12: Example behavior of the CAML controller with PI structure when tracking a 0.1 m/s airspeed step, over four episodes. The softmax output probabilities (p_i) of each gain set, and therefore the gains themselves, change rapidly, especially during the beginning of each episode. Changes occur less often during later episodes. Performance, measured with J, does not necessarily improve over time.

inappropriate gain set even if the perfect model is known. In spite of this limitation, CAML has a tendency to improve over time. Note that the gains selected during the final episodes in figure 12 are close to the optimal gains found by the Policy Gradient controller in figure 4.

V. Results and Discussion

This section treats the results of the test problems mentioned in section II. Results are shown for the Policy Gradient algorithm and for CAML. Additionally, results for a fixed PI controller are shown. All gain sets from the CAML database are attempted in a fixed PI controller, and the best performing set is shown.

V.A. Baseline tracking test

Figure 13 shows the performance of the algorithms during the baseline tracking test, where the goal of the controller is to track a 0.1 m/s velocity step on all of the 46 available models. The score S is defined in Eq. 14. Wilcoxon's signed rank test[24] is used to assess the statistical significance of the differences in tracking accuracy[i], rather than score. This test works in a pairwise setting; the p-values of several pairs are shown in

[i]This test assumes a zero hypothesis that all data comes from the same type of continuous distribution, which is clearly false if unstable models are included. All models with one or more unstable controllers are therefore disregarded in this statistical test. This means that the test only considers tracking accuracy, not the combined score S.

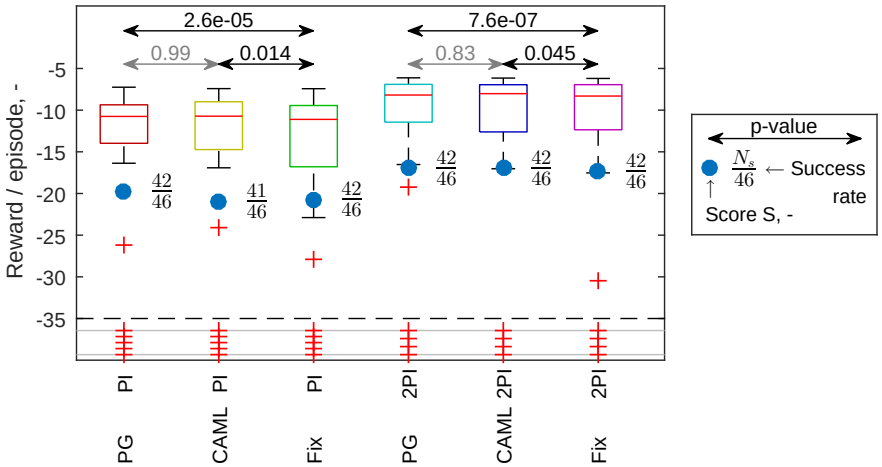


Figure 13: Results of the baseline tracking test. The statistical significance is shown for some differences in tracking accuracy. The Machine Learning algorithms perform better overall.

figure 13. The difference is considered significant if $p < 0.05$.

$$S = -100 \cdot N_{unstable} + \sum_{i=1}^{N_{stable}} J_i, \qquad J = \sum_t U(t) = -\sum_t \left[(V(t) - 0.1)^2 + q(t)^2\right] \tag{14}$$

It can be seen that all algorithms manage to stabilize at least 41 out of 46 models. When inspecting the single loop PI variants of the ML algorithms, the score of the PG algorithm is slightly higher, but the increase in tracking accuracy is not significant. Both algorithms outperform the fixed PI controller in terms of tracking accuracy. The initial gains of the PG algorithm are the same as the gains of the fixed PI controller. The fact that the success rates are equal, shows that the variable learning rate procedure is effective in avoiding instability.

The single loop variant of CAML has a lower success rate than the two other approaches. Further inspection shows that four of the 46 linear DelFly models cannot be controlled successfully by a single loop PI controller of the given structure. Furthermore, there is a universal set of gains, which results in success for the 42 remaining models. By using this gain set as the initial setting, this allows the PG algorithm to successfully control those 42 models. CAML cannot improve safety with its rapid gain changes, because changes from the universal gain set will not allow control of the four 'uncontrollable' models. It therefore only experiences its disadvantage, which is an occasionally incorrect gain selection. This results in instability of one of the 42 controllable models.

For cascaded PI controllers, the performance of both ML algorithms looks comparable to the performance of the fixed controller in figure 13. However, the tracking accuracy of the fixed controller is actually significantly worse, due to the distribution of the outliers in the figure. Cascaded PI controllers lead to higher performance than single loop PI controllers, for all algorithms considered. All algorithms allow control of 42 out of 46 models. Just as with the single loop variant, there is a universal gain set for those 46 models. Four models (the same four as before) cannot be controlled successfully by cascaded PI controllers.

V.B. Noise

The robustness of the algorithms to measurement noise on the states is presented in figure 14. Overall, it seems that increasing noise does not lead to a dramatic decrease in performance. The expected noise level of the DelFly does not lead to problems.

CAML is more sensitive to noise than the PG algorithm. This can be explained by looking at the way in which noise affects the algorithms. In the PG algorithm, measurement noise changes the time-varying part of the internal model, leading to a disturbance signal in the virtual gradient simulation. Since this disturbance is random, its effect is diminished by using longer measurement times. For CAML, noise gives problems with model identification. As explained in section IV, Recursive Least Squares identification cannot deal

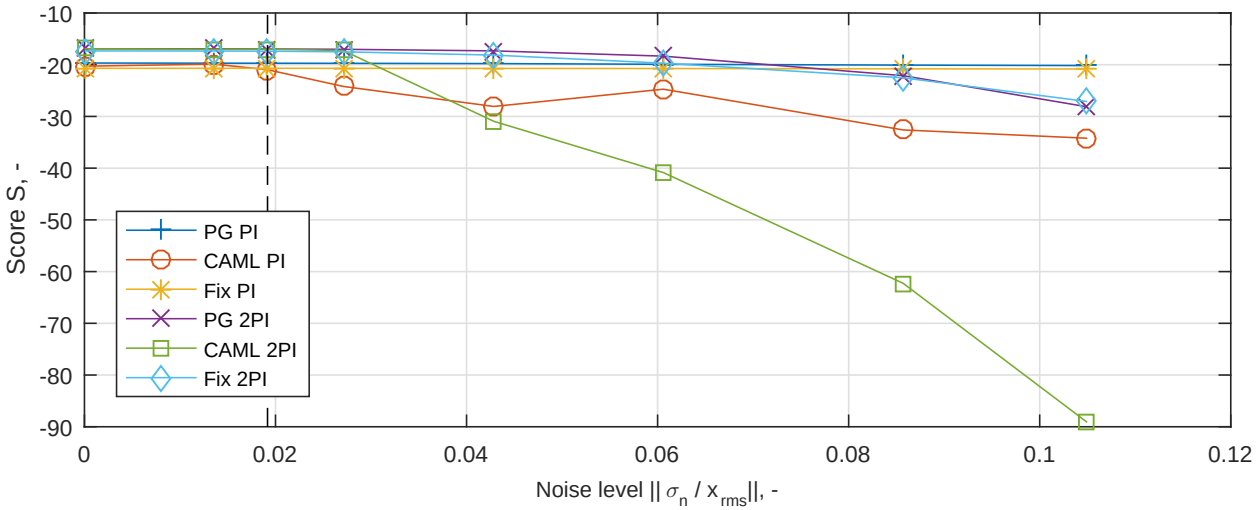


Figure 14: Sensitivity of the algorithms to noise. The PG algorithm is more robust to noise, with the single loop variant performing better.

with noisy measurements of the current state. The RML1 algorithm solves this problem to some extent, but increased noise will still affect the identified model.

Another noteworthy conclusion is that the algorithms with cascaded PI controllers are more sensitive to measurement noise than the algorithms with single loop PI controllers. This seems to be due to the low-level control, because the fixed PI controllers show the same effect. Because cascaded controllers make use of two states, the noise on both states enters the PI controller, lowering the accuracy.

V.C. Disturbances

An unknown disturbance on the elevator will strongly affect the performance of the low-level PI controllers. Also, the algorithms will perform suboptimally. It is expected that the algorithms can deal with disturbances more easily than with noise, because the disturbance actually changes the state. The PG algorithm will sample the disturbance on each step, and will find the correct gradient for a simulation with this disturbance sequence. The model identification algorithm in CAML can deal with disturbances directly, since they result in an actual state change.

Figure 15 shows the performance of the algorithms with a varying level of disturbances. The robustness of the PG algorithm is found to be stronger than the robustness of CAML, because the algorithm can actively deal with the disturbances by moving to a more appropriate gain setting. CAML lacks this option.

In this test, it is found that cascaded PI controllers result in better scores than single loop PI controllers. This makes sense, because the double loop controllers can counteract disturbances on two states. An offset in pitch angle will eventually lead to a speed change (which is punished), and only the double loop controllers can counteract this.

V.D. Nonlinearity

When nonlinearities are introduced, the linear controllers will become less effective. Figure 16 shows that the nonlinearities considered are mild enough to allow accurate control with fixed PI controllers. The nonlinearity metric ϕ is defined in the appendix.

This analysis shows that the PG algorithm is more robust to nonlinearities than CAML. This was expected, because CAML requires identification of a linear model, which may be inaccurate on a nonlinear system. For the Policy Gradient approach, the nonlinearity is dealt with like a linear model deviation.

Double loop PI controllers are able to achieve higher performance than single loop controllers. The explanation for this is that cascaded controllers limit oscillations of the pitch angle, and thereby keep the system closer to its trimmed point. This limits the effect of the nonlinearity.

It is noticed that with a fixed cascaded PI controller, the score increases for high nonlinearity. This is an artifact of the specific nonlinearity that is introduced. If the linearized systems are compared to the

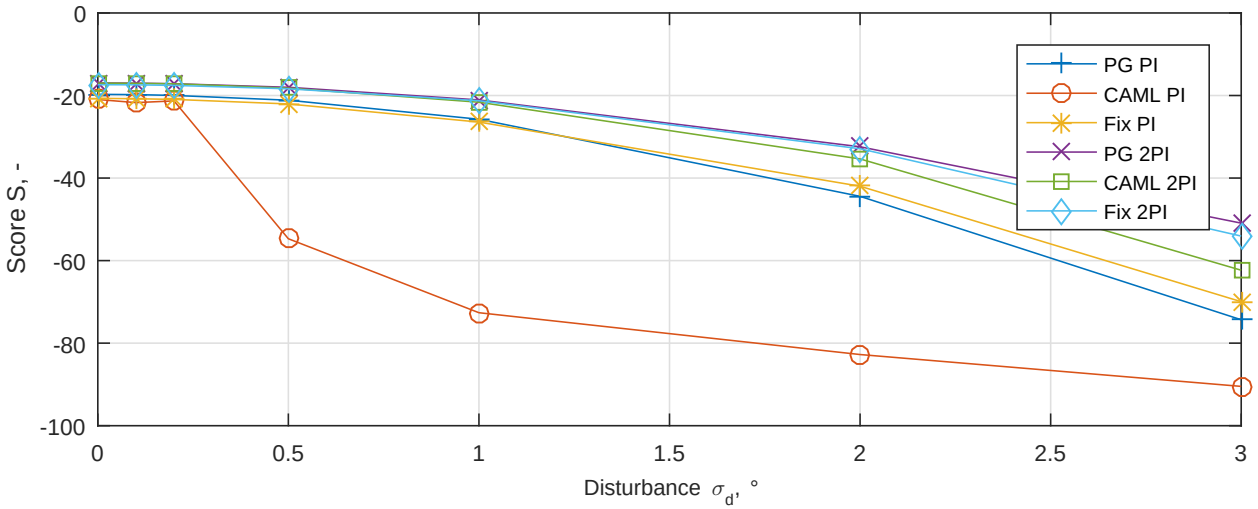


Figure 15: Sensitivity of the algorithms to disturbances. The PG algorithm is more robust to noise, with the double loop variant performing better.

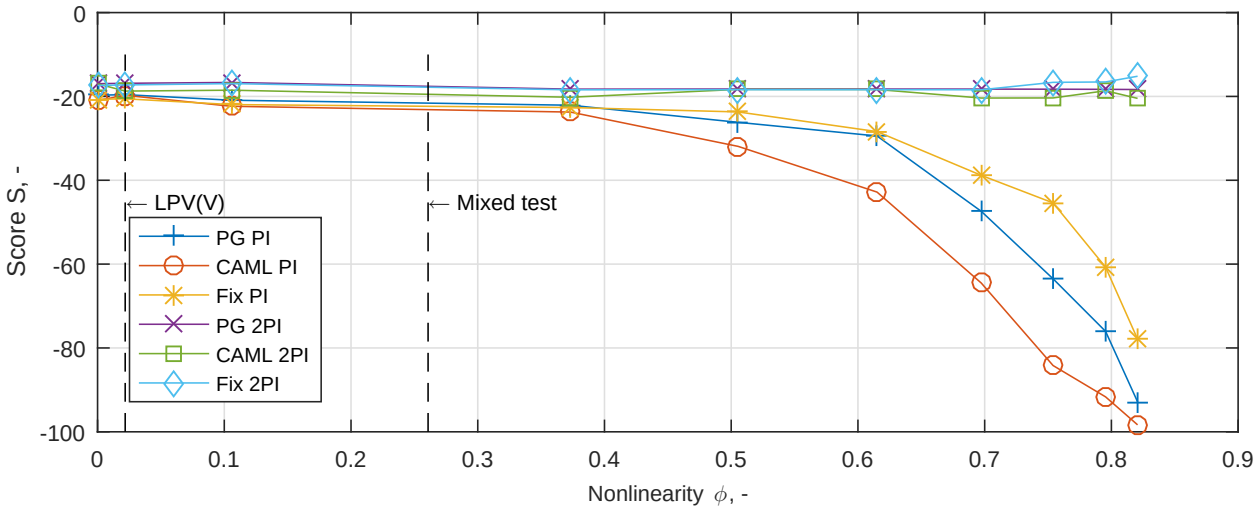


Figure 16: Sensitivity of the algorithms to nonlinearity (defined in the appendix). The PG algorithm is more robust to nonlinearity, with the double loop variant performing better.

linearized systems at a 0.1 m/s airspeed, it is found that the higher airspeed typically increases stability. Likewise, a lower airspeed often results in instability: if a -0.1 m/s step is taken, the nonlinear systems become more difficult to control.

V.E. Flapping

When a flapping motion is introduced, the performance of all algorithms suffers, as shown in figure 17. This figure also shows the performance during a simulation where the filter is installed, but flapping is not added. The flapping can be considered a disturbance, which cannot be filtered away completely. This affects the low-level PI controllers, as well as the learning processes. The time-varying model of the PG algorithm will be altered, which has an effect on the gradient. CAML suffers by identifying an incorrect model, leading to an incorrect gain selection. The effect of flapping on CAML seems to be significantly stronger. The PG algorithm has a tracking accuracy comparable to a fixed PI controller.

For both algorithms, the performance of the cascaded controllers is degraded severely compared to the non-flapping simulations, which seems to conflict with the high performance of the fixed cascaded PI controller. A closer look at the individual fixed PI controllers in figure 18 provides an explanation. For the

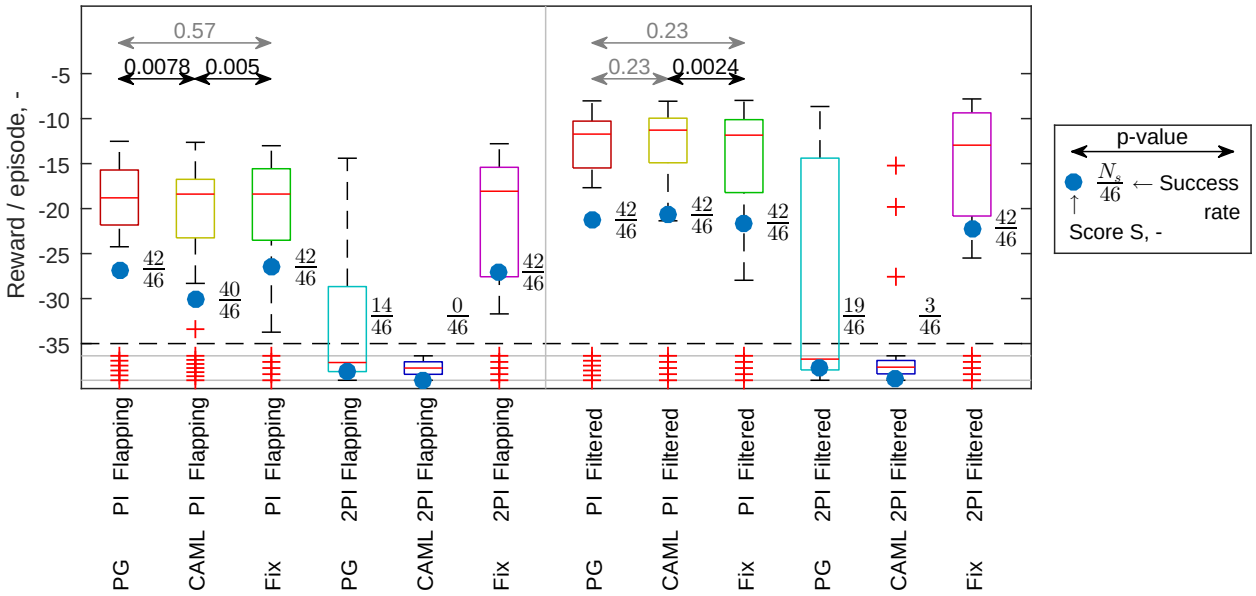


Figure 17: Results of the flapping test, compared to a test where the states are filtered, but without flapping. The statistical significance is shown for some differences in tracking accuracy. CAML performs worse than the other algorithms if flapping is present. Cascaded intelligent PI controllers perform worse with flapping.

single loop PI controllers in figure 18a, the performance of each controller degrades to a comparable extent. The cascaded PI controllers in figure 18b, however, show a varying level of degradation. Most of the gain sets show large deterioration of performance, but some are robust. Because the algorithms do not account for flapping directly, they cannot anticipate this, resulting in incorrect gain selections. This is especially true for CAML.

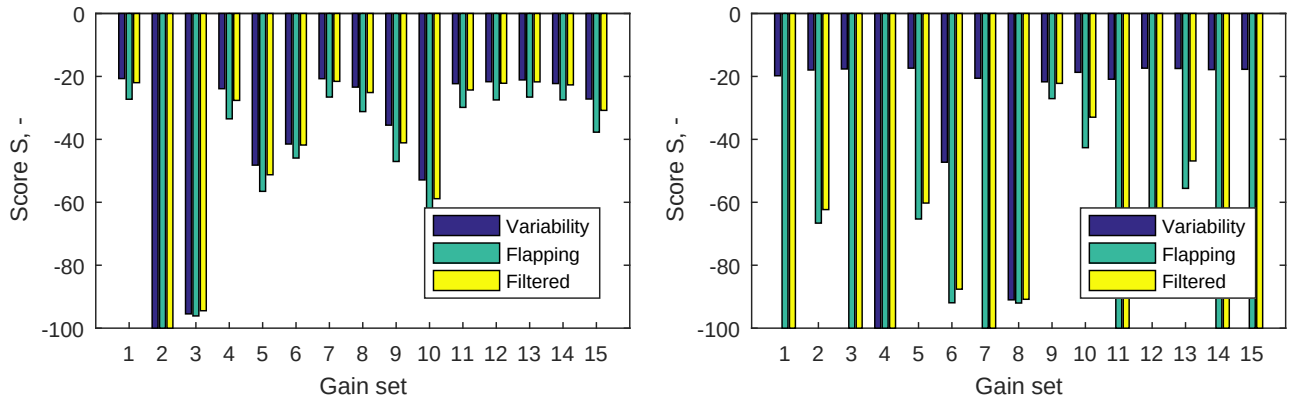
The fact that the performance with filter, but without flapping is also low, proves that this is due to the filtering process. The cascaded PI controllers, operating at higher frequencies, are more sensitive to this. Figure 19 shows the Bode diagram of the inner open loop (pitch angle control) with gain sets 1 and 9. Because the phase margin with gain set 1 is very small, it will become unstable if a phase lag is introduced by the filter.

V.F. Mixed test

In the final test, the noise, nonlinearity, and flapping are considered at the same time. Note that this test does not consider 46 different models, but only one model at 46 trim points. The results shown in figure 20 are therefore not directly comparable to the previous simulations, which also accounted for variability.

The single loop PG algorithm performs significantly better than CAML on the mixed test. CAML is affected more by both nonlinearity and flapping, resulting in lower performance. The difference between the PG algorithm and a fixed PI controller is not significant.

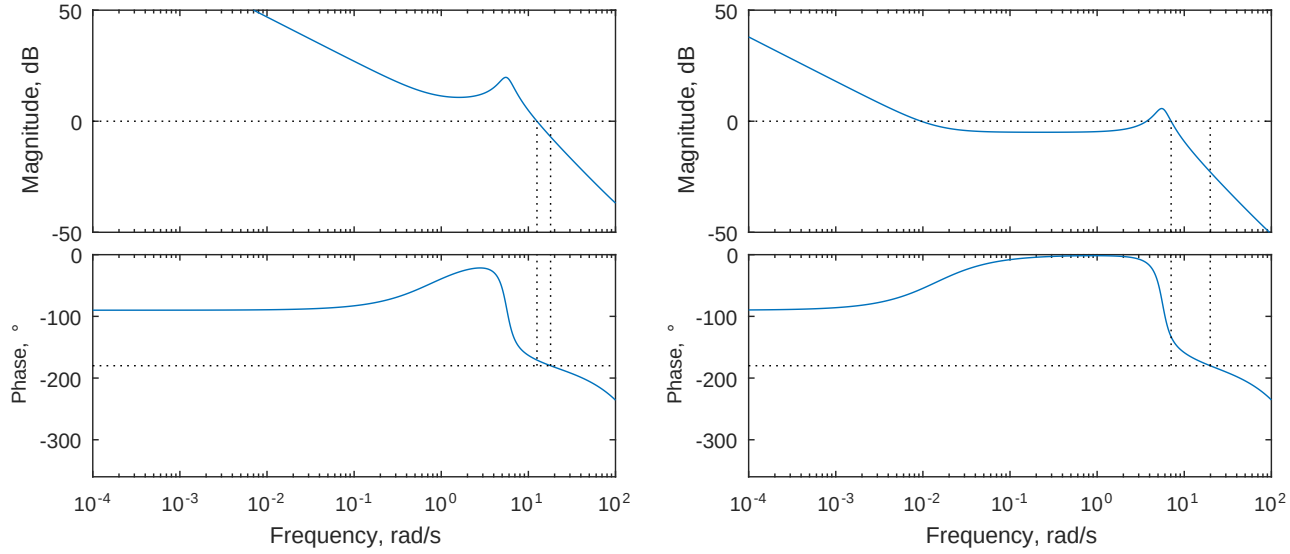
The cascaded versions of the algorithms are both showing lower performance. This is consistent with the findings of the flapping tests. It is therefore expected that this is caused by the filtering effect. It is again found that this effect is larger for CAML.



(a) Single loop PI

(b) Cascaded PI

Figure 18: Performance of the individual fixed PI controllers on the variability and flapping tests, and with filter only. Some of the cascaded PI controllers are very sensitive to the filtering, but the flapping itself has little influence.



(a) Gain set 1 (sensitive)

(b) Gain set 9 (insensitive)

Figure 19: Bode diagrams of the inner loop of an example DelFly with filter-sensitive gain set 1 and insensitive gain set 9. The small gain margin with gain set 1 becomes problematic if the filter is added.

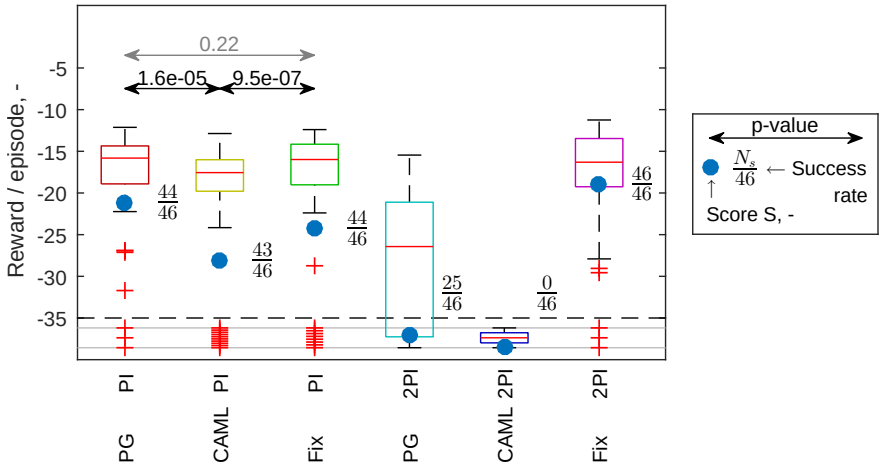


Figure 20: Results of the mixed test, where only one model is considered. CAML performs worse than the other algorithms.

VI. Conclusion and Recommendations

The DelFly is a difficult vehicle to control because of its flapping dynamics and nonlinearity. Also, there is variability between different individual vehicles, measurements are corrupted by noise, and disturbances change the state. Two Machine Learning control algorithms were used to tackle these challenges. First, a Policy Gradient algorithm of Reinforcement Learning is used, where the gradient is based on an inaccurate model. This approach is extended with a new variable learning rate technique in order to improve safety.

Second, CAML is proposed as a new algorithm to tackle such problems. This approach identifies a model, which is used to select the most appropriate PI gain set with a NN classifier. The selectable gain sets are predefined. This algorithm can change the gains rapidly during flight, which may improve safety. It is shown that identification of a stochastic model is not necessary. The RML1 algorithm is used to lower the effect of state noise during model identification, but this only works without flapping motion.

A baseline test, with 46 DelFly models, a fixed noise level, and no flapping dynamics, shows that the Policy Gradient approach results in more accurate tracking than a fixed PI controller. The PG algorithm has a tracking accuracy comparable to CAML, but this algorithm results in a lower safety on the test problem. This is because there is a universal gain set, which is able to control 42 out of 46 models. The remaining four models cannot be controlled by PI controllers of the selected structure, which means that CAML is also unable to control these, and cannot obtain a safety increase by its rapid gain changes. The PG algorithm is recommended for a problem with a universal gain set. It is recommended to test CAML on a problem without a universal gain set, and to ensure that all models can be controlled by at least one of the gain sets.

The PG algorithm has a similar robustness to noise as the predefined controller. CAML is more sensitive to noise, because this complicates model identification. Single loop CAML is more sensitive to disturbances than the other algorithms, and shows a fast decrease in performance. Single loop CAML is therefore not recommended for problems with high disturbance levels. The other algorithms perform comparably.

When flapping is introduced, single loop CAML performs slightly worse than a fixed PI controller, while the PG algorithm performs obtains a performance similar to a fixed controller. A low-pass filter is required, such that control occurs only at lower frequencies. This results in problems with the cascaded ML algorithms, because many combinations of models and gain sets have a low phase margin and become unstable due to the phase lag of the filter. It is recommended to test other types of filters, in order to improve real-time control. The Policy Gradient approach could be improved by explicitly using the filter to select the gradient direction. If unfiltered states are sampled, and the filter is applied afterwards, one can find a gradient direction accounting for filtering. CAML can be improved by accounting for filtering in the gain set generation, such that the selection takes into account filtering effects.

On a mixed test involving flapping, nonlinearity and noise, it is found that the PG algorithm performs best. CAML cannot match the performance of a fixed PI controller. Due to the filter, cascaded ML controllers show low performance. The filtering method should be improved for effective use of cascaded controllers.

The CAML algorithm may be improved by letting the experienced reward influence selection. It was found that model identification is difficult in the presence of filtering. It is possible to improve the model after each episode, by storing the states and applying the zero-phase filter offline. The selection process itself may be improved by testing other function approximators, e.g., deep NNs. The computational load may be alleviated by evaluating the classifier at a lower frequency than the low-level PI controller. A thorough analysis of computational time is recommended for future research.

A new algorithm that follows naturally from this research combines the CAML and PG algorithms sequentially. One may imagine a controller that uses CAML for the first instances, until the gain set remains constant for some time. Then, the gains can be fine-tuned using the PG algorithm. Such a method may improve both safety and tracking accuracy. Research into such a method is therefore recommended.

This research was limited to single or double loop PI controllers. However, literature has demonstrated the use of pitch (rate) dampers[3] and dynamic inversion-based concepts.[5] It should be investigated whether the ML algorithms can be implemented for such controllers as well. Another limitation is that this paper has only treated elevator control. New models are required to investigate the control of the flapping frequency and rudder.

Proving the effectiveness of the algorithms on simulation models is limited by the validity of the models of the dynamics, noise and disturbances. A final controller should be subjected to a physical flight test for validation. This paper is a step towards such a flight test. Promising results were obtained with the Policy Gradient algorithm. It is suggested to use this algorithm, with the newly developed variable learning rate technique, for further development of a DelFly controller.

Appendix

This section demonstrates how the nonlinearity metric is defined. An artificial nonlinearity is superposed on the linear system. This nonlinearity is based on the Linear Parameter-Varying (LPV) system in Eq. 15.

$$x(t+1) = (A_0 + A_v V)x(t) + (B_0 + B_v V)\delta_e(t) \tag{15}$$

The parameters of the A- and B-matrices are assumed to be linearly dependent on the airspeed. Although the nonlinearity is artificial, it is based on the 46 models, by examining the variation of the aerodynamic parameters with the trimmed airspeeds of the models. Figure 21 shows the variation of three aerodynamic parameters with airspeed.

The LPV model can be estimated by performing linear regression on the data in figure 21. This is performed with a univariate linear function for all aerodynamic parameters, resulting in the A_0-, A_v-, B_0- and B_v-matrices. These matrices form an unvalidated LPV model of the DelFly. Validation of this model is not the focus of this work. However, the A_v- and B_v matrices provide an indication of the level of nonlinearity of the model. A scaled version of the nonlinear part of Eq. 15 is superposed on all 46 linear models, as shown in Eq. 16.

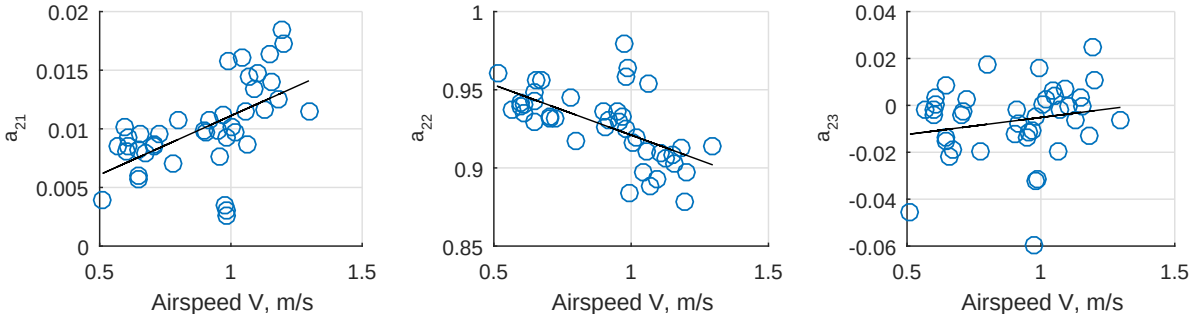
$$x_i(t+1) = (A_i + \Lambda A_v(V - V_{0,i}))x(t+1) + (B_i + \Lambda B_v(V - V_{0,i}))\delta_e(t) \tag{16}$$

By varying the scaling parameter Λ, and repeatedly performing the variability analysis, the sensitivities of the algorithms to nonlinearity can be analyzed. With $\Lambda = 0$, the 46 local linear models are obtained, while $\Lambda = 1$ corresponds to the expected level of nonlinearity.

While Λ provides an indication of the level of nonlinearity, it is only applicable to models of the form of 15. A more general metric is required in order to allow the use of other model structures. A generic metric[25] is based on the difference between the linear system and the nonlinear system. It requires the definition of a nonlinear system N, with $y_N = N[u, x_{N,0}]$, and a linear system G, with $y_G = G[u, x_{G,0}]$. The y and u are signals, with a time dimension. The operators N and G map the scalar input signal u to the vector output signal y, starting from initial conditions $x_{N,0}$ and $x_{G,0}$. The spaces $\mathcal{U}_a$ and $\mathcal{Y}_a$ are defined as the allowable spaces of the signals u and y. Furthermore, the spaces $\mathcal{X}_{0,a}$ and $\mathcal{X}_{0,G}$ are the allowable spaces of the initial states of the nonlinear and linear systems, respectively. Finally, $\mathcal{G}$ is the space of allowable linear systems. The nonlinearity measure used in this research[25] is presented in Eq. 17.

$$\phi(t_f) = \inf_{G\in\mathcal{G}} \sup_{u,x_{N,0}\in\mathcal{S}} \inf_{x_{G,0}\in\mathcal{X}_{0,G}} \frac{\|G[u, x_{G,0}] - N[u, x_{N,0}]\|_y}{\|N[u, x_{N,0}]\|_y} \tag{17}$$
$$\text{with } \mathcal{S} = \{(u, x_{N,0}) :\ u \in \mathcal{U}_a,\ x_{N,0} \in \mathcal{X}_{0,a},\ N[u, X_{N,0}] \in \mathcal{Y}_a\}$$

The measure ϕ takes a suitable norm of the difference signal between N and G, and normalizes it by the norm of the nonlinear output signal. This is a complex optimization problem, because it requires



(a) a_{21} (b) a_{22} (c) a_{23}

Figure 21: Generation of the artificial nonlinearity for three aerodynamic parameters. Aerodynamic parameters of the 46 models are shown, together with the estimated trend.

optimization over all allowable linear models, with all allowable initial linear states. This is done for the worst-case scenario of the input signal and initial nonlinear state, and requires the norm of a signal from $t = 0$ to $t = t_f$. Directly calculating the solution of this problem is practically infeasible.[25]

In this research, the problem is solved by limiting the allowable spaces of the signals and states to a single point. For each of the 46 models, there is a trimmed state $x_{0,i}$. The limitation is enforced that $x_{N,0} = x_{G,0} = x_{0,i}$ for each of the models. Furthermore, this paper only considers one allowable input signal u_s, which is a 0.1 rad elevator step. If the responses of N and G to this step are similar, it is expected that the models will be similar in general. Finally, the 'best' linear model is fixed; the 46 linear models F provide a reasonable approximation of the optimal linear model[j]. The final time is set to $t_f = 1$ s. This removes all infimum and supremum operators, arriving at Eq. 18 for the nonlinearity metric used in this research.

$$\phi_i = \frac{\|F\left[u_s, x_{0,i}\right] - N\left[u_s, x_{0,i}\right]\|_y}{\|N\left[u_s, x_{0,i}\right]\|_y} \tag{18}$$

The norm that is used represents the RMS values of the signals, where each state is scaled by the reference RMS value used for the noise metric. This norm is presented in Eq. 19.

$$\|y\|_y = \sqrt{\sum_{t=0}^{t_f} \sum_{i=1}^{4} \left(\frac{y_i}{y_{i,ref}}\right)^2} \tag{19}$$

The metric ϕ_i is calculated for each of the 46 models. For each Λ, the median of the 46 ϕ_i is used as the nonlinearity.

Acknowledgments

The authors would like to thank Frank Rijks for developing and demonstrating the DelFly in figure 1.

References

[1]De Croon, G. C. H. E., Perçin, M., Remes, B. D. W., Ruijsink, R., and De Wagter, C., *The DelFly: Design, Aerodynamics and Artificial Intelligence of a Flapping Wing Robot*, Springer Netherlands, 1st ed., 2016.

[2]De Croon, G. C. H. E., Groen, M. A., De Wagter, C., Remes, B. D. W., Ruijsink, R., and Van Oudheusden, B. W., "Design, Aerodynamics and Autonomy of the DelFly," *Bioinspiration & biomimetics*, Vol. 7, No. 2, May 2012.

[3]De Wagter, C., Koopmans, J. A., De Croon, G. C. H. E., Remes, B. D. W., and Ruijsink, R., "Autonomous Wind Tunnel Free-Flight of a Flapping Wing MAV," *Advances in Aerospace Guidance, Navigation and Control: Selected Papers of the Second CEAS Specialist Conference on Guidance, Navigation and Control*, edited by Q. P. Chu, J. A. Mulder, D. Choukroun, E. Van Kampen, C. De Visser, and G. Looye, April 2013, pp. 603–621.

[4]Verboom, J. L., Tijmons, S., De Wagter, C., Remes, B. D. W., Babuška, R., and De Croon, G. C. H. E., "Attitude and Altitude Estimation and Control on board a Flapping Wing Micro Air Vehicle," *IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5846–5851.

[5]Cunis, T., Karásek, M., and De Croon, G. C. H. E., "Precision Position Control of the DelFly II Flapping-wing Micro Air Vehicle in a Wind-tunnel," *International Micro Air Vehicle Conference and Competition (IMAV)*, Oct. 2016.

[6]Junell, J. L., Mannucci, T., Zhou, Y., and Van Kampen, E., "Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning," *AIAA Guidance, Navigation, and Control Conference*, Jan. 2016.

[7]Caetano, J. V., De Visser, C. C., De Croon, G. C. H. E., Remes, B. D. W., De Wagter, C., Verboom, J., and Mulder, M., "Linear Aerodynamic Model Identification of a Flapping Wing MAV based on Flight Test Data," *International Journal of Micro Air Vehicles*, Vol. 5, No. 4, Dec. 2013, pp. 273–286.

[8]Armanini, S. F., De Visser, C. C., De Croon, G. C. H. E., and Mulder, M., "Time-Varying Model Identification of Flapping-Wing Vehicle Dynamics Using Flight Data," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 3, March 2016, pp. 526–541.

[9]Chang, J., Armanini, S. F., and De Visser, C. C., "Feasibility of LTI-Model-Based LPV Model of DelFly," Unpublished MSc paper.

[10]Van Kampen, E., Chu, Q. P., and Mulder, J. A., "Continuous Adaptive Critic Flight Control Aided with Approximated Plant Dynamics," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Aug. 2006.

[11]Sutton, R. S. and Barto, A. G., *Reinforcement learning: An introduction*, MIT press, 1st ed., 1998.

[j]This approximation is not perfect, because the step input causes the state to vary. The real optimal linear model would have a different trimmed condition, but is difficult to determine.

[12]Motamed, M. and Yan, J., "A Reinforcement Learning Approach to Lift Generation in Flapping MAVs: Experimental Results," *IEEE International Conference on Robotics and Automation (ICRA)*, April 2007, pp. 748–754.

[13]Motamed, M. and Yan, J., "A Reinforcement Learning Approach to Lift Generation in Flapping MAVs: Simulation Results," *IEEE International Conference on Robotics and Automation (ICRA)*, May 2006, pp. 2150–2155.

[14]Roberts, J. W., Moret, L., Zhang, J., and Tedrake, R., "Motor Learning at Intermediate Reynolds Number: Experiments with Policy Gradient on the Flapping Flight of a Rigid Wing," *From Motor Learning to Interaction Learning in Robots*, edited by O. Sigaud and J. Peters, Vol. 264 of *Studies in Computational Intelligence*, Springer Berlin Heidelberg, 2010, pp. 293–309.

[15]Wang, J. and Kim, J., "Optimization of Fish-like Locomotion using Hierarchical Reinforcement Learning," *International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Oct. 2015.

[16]Abbeel, P., Quigley, M., and Ng, A. Y., "Using Inaccurate Models in Reinforcement Learning," *International Conference on Machine Learning (ICML)*, June 2006.

[17]Armanini, S. F., Karásek, M., De Visser, C. C., De Croon, G. C. H. E., and Mulder, M., "Flight Testing and Preliminary Analysis for Global System Identification of Ornithopter Dynamics using On-board and Off-board Data," *AIAA Atmospheric Flight Mechanics Conference*, Jan. 2017.

[18]Lupashin, S., Schöllig, A., Sherback, M., and D'Andrea, R., "A Simple Learning Strategy for High-speed Quadrocopter Multi-flips," *IEEE International Conference on Robotics and Automation (ICRA)*, May 2010, pp. 1642–1648.

[19]Koppejan, R. and Whiteson, S., "Neuroevolutionary Reinforcement Learning for Generalized Control of Simulated Helicopters," *Evolutionary Intelligence*, Vol. 4, No. 4, Dec. 2011, pp. 219–241.

[20]Moldovan, T. M., Levine, S., Jordan, M. I., and Abbeel, P., "Optimism-driven Exploration for Nonlinear Systems," *IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3239–3246.

[21]Arthur, D. and Vassilvitskii, S., "k-means++: The Advantages of Careful Seeding," *ACM-SIAM Symposium on Discrete Algorithms*, Jan. 2007.

[22]Gustavsson, I., Ljung, L., and Söderström, T., "A Comparative Study of Recursive Identification Methods," Tech. Rep. TFRT; Vol 3085, Department of Automatic Control, Lund Institute of Technology (LTH), Jan. 1974.

[23]Friedlander, B., "System Identification Techniques for Adaptive Signal Processing," *Circuits, Systems and Signal Processing*, Vol. 1, No. 1, 1982, pp. 3–41.

[24]Wilcoxon, F., "Individual Comparisons by Ranking Methods," *Biometrics bulletin*, Vol. 1, No. 6, Dec. 1945, pp. 80–83.

[25]Helbig, A., Marquardt, W., and Allgöwer, F., "Nonlinearity Measures: Definition, Computation and Applications," *Journal of Process Control*, Vol. 10, No. 2, April 2000, pp. 113–123.