Self-tuning gains of a quadrotor using a simple model for policy gradient reinforcement learning

Junell, JL; Mannucci, T; Zhou, Y; van Kampen, EJ

DOI
10.2514/6.2016-1387

Publication date
2016

Document Version
Accepted author manuscript

Published in
Proceedings of the AIAA guidance, navigation, and control conference

Self-tuning Gains of a Quadrotor using a Simple Model for Policy Gradient Reinforcement Learning

Jaime Junell*, Tommaso Mannucci*, Ye Zhou*, and Erik-Jan van Kampen†
Delft University of Technology. Delft, The Netherlands

February 28, 2017

Abstract

Autonomous flight of Micro Aerial Vehicles (MAVs) faces many challenges in the realm of control. When approaching these problems it is usually advantageous to have an accurate mathematical model of the system to be controlled. However, this is not always possible to obtain due to the complex nature of MAV dynamics, inconsistency of manufacturing processes, and parameter adaptation over time. A model-free control approach such as model-free Reinforcement Learning (RL) requires a large number of real-life trials to learn desired actions. This paper presents the implementation of an approach which utilizes a simple model of a quadrotor system to reduce the number of real-life trials required to find the locally optimal gains. The concept is verified using 2 simulations: one for an F-16 aircraft and the other for a quadrotor. Finally, the approach is applied on a quadrotor's vertical controller to tune the PID gains during a takeoff maneuver of a real AR.drone 2 quadrotor.

Nomenclature

MDP	Markov decision process
π_θ	policy
θ	policy parameters
ρ	performance measure
V^π	value function of policy, π
$\nabla\theta\rho(\theta)$	derivative vector of $\frac{\delta\rho}{\delta\theta}$

1 Introduction

Autonomous flight of Unmanned Aerial Vehicles(UAVs) and, especially, Micro Aerial Vehicles (MAVs) is challenging in the realm of control for many reasons. Dynamics of the system can be difficult to model, the vehicles are often more susceptible to unforeseen disturbances, and they can have sensor or actuator limitations due to weight and size requirements. One method to address these problems is to create learning or adaptive controllers. Reinforcement learning (RL), for example, can make a system learn desired actions with little or no a priori knowledge of its dynamics or environment and can adapt to changing conditions [1]. For these reasons, RL has become a promising tool for improving autonomous flight in many different types of UAVs and MAVs [2, 3, 4]. This paper will focus on hybrid approach which uses a simple or incomplete model of a system in order to decrease the number of real-life trials needed in reinforcement learning policy improvement.

Policy improvement for a finite number of states and actions is an attractive method for policy search since it guarantees convergence to the optimal policy, π^* as well as an optimal value function, V^* for all states [1]. However, sweeps of the whole state set are required each iteration. Modern computers can

*PhD Student. TU Delft Aerospace Engineering, Control and Simulation. Delft, The Netherlands. AIAA Member Grade.

†Assistant Professor. TU Delft Aerospace Engineering, Control and Simulation. Delft, The Netherlands. AIAA Member Grade.

now handle the computational load of large state sets but of course there are limits and methods have been explored to cope with high dimensionality [5, 6]. For continuous problems, function approximations have been used with success [7, 8]. Guarantees of local optimum policies are still present but a near-perfect model of a system is necessary. By using a crude model in simulation to do some of the heavy computational work, it is possible to decrease the number of real-life trials and still guarantee convergence to a locally optimal policy [9, 10].

This hybrid model-based approach is advantageous for vehicles which need local tuning of gains, since it is guaranteed to find the local optimum but not the global. The AR.drone 2 quadrotor could benefit from this since the blades of the rotors can become less efficient over time, therefore decreasing the maximum amount of thrust available or adding additional actuator delay. An optimal policy can be found and used by some means but will shift over time. By using the old optimal gains as a starting point, the new optimum can be found with this approach. Another vehicle which could benefit is a flapping wing MAV such as the Delfly [1] [2]. Efforts are being made to model this complex system [11, 12], however inconsistencies in the manufacturing make each individual vehicle unique. Hand tuning the gains of the controller gives desirable performance, however it is time consuming work. This approach could use the finely tuned gains from, say, vehicle A as a starting point, and as long as the newly manufactured vehicle B is not too different from A, the optimal gains for this new vehicle should be reachable with only a few real-life trials using a "good enough" model and policy gradient RL.

A background of reinforcement learning policy iteration and other necessary information as it is applied to this problem is outlined in Section 2. The simulation setup and results will first be presented in Section 4 before the application setup and results in Section 5. An overview of all the results is discussed in Section 6.

2 Background

Background information will now be presented on policy improvement reinforcement learning and the hybrid approach which inspired this research.

2.1 Gradient Policy Iteration in Reinforcement learning

The basic framework of RL consists of a learning agent and its interaction with a finite Markov decision process(MDP). Let S and $A(s)$ be defined as finite sets of states and actions, respectively. $P(s'|s,a)$ is the state transition probability from state s to s' given action a, and R is the reward function. The policy, π, is a mapping between states and actions. The value function, $V^\pi(s)$, represents the value of being in state s given that policy π is being followed. The goal of reinforcement learning, in general, is to gain information about V^π using interaction with the environment in order to find an optimal policy from which to determine desired actions. Policy iteration, iteratively improves upon V^π and π in phases as seen in Equation 1 [1]. Where $\overset{E}{\rightarrow}$ denotes an evaluation phase of the policy to obtain $V^\pi(s)$ for all $s \in S$, and $\overset{I}{\rightarrow}$ denotes a policy improvement phase in which $\pi(s)$ is found for all $s \in S$.

$$\pi_0 \overset{E}{\rightarrow} V^{\pi_0} \overset{I}{\rightarrow} \pi_1 \overset{E}{\rightarrow} V^{\pi_1} \overset{I}{\rightarrow} \pi_2 \overset{E}{\rightarrow} ... \overset{I}{\rightarrow} \pi^* \overset{E}{\rightarrow} V^* \tag{1}$$

Since every state in the finite state set is evaluated each iteration, this approach is guaranteed to converge to the global optimum, V^* and π^*. However, one can see how a large state set can lead to an unmanageable amount of computation to reach this optimum. Additionally, this guarantee only applies to discrete cases where as most real world problems are in the continuous domain. Therefore, this concept can be used with function approximation for the value prediction in order to alleviate the number of necessary evaluation computations in a continuous state problem. A simple and widely used parameter update method for function approximation is gradient-descent. In this case, the policy, π_θ is represented by parameters $\vec{\theta_t} = (\theta_t(1), \theta_t(2), ...\theta_t(N))$, and V_t is any differentiable function of $\vec{\theta_t}$.

$$V_t = f(\vec{\theta_t}) \tag{2}$$

With this, one can setup a parameter update rule:

[1] TU Delft Robotics Institute. http://robotics.tudelft.nl

[2] http://www.delfly.nl/index.html

$$\vec{\theta_{t+1}} = \vec{\theta_t} - \alpha \nabla_{\vec{\theta_t}} f(\vec{\theta_t}) \tag{3}$$

Where α is the stepsize and $\nabla_{\vec{\theta_t}} f(\vec{\theta_t})$ denotes the partial derivative vector for any function f,

$$\nabla_{\vec{\theta_t}} f(\vec{\theta_t}) = \left(\frac{\delta f(\vec{\theta_t})}{\delta \theta_t(1)}, \frac{\delta f(\vec{\theta_t})}{\delta \theta_t(2)}, ..., \frac{\delta f(\vec{\theta_t})}{\delta \theta_t(N)} \right) \tag{4}$$

The function f can also be denoted as a performance measure, ρ. As for the performance measures in this paper, the integral absolute error is used for the F-16 simulation and the mean absolute error (MAE) is used for the quadrotor. The derivative vector will point in the direction the error decreases most rapidly.

This approach has been proven [8] to converge to a *local* optimal policy for the performance measure [3].

Even though the local optimum is guaranteed, and even though it applies to continuous space using a function approximation, the optimal policy is solved for the *model* of the true system, with which the performance measure is evaluated. If the policy is learned on a model and intended for use on a real system, the guarantee is no longer valid unless the model is a perfect representation. Otherwise, the optimal policy can be learned on the real system if that is an option, but to find a crude derivative vector will require at least as many real-life trials as number of parameters in $\vec{\theta_t}$ for *each* iteration.

What if a perfect model is not available? Can this method still be used? The answer to these questions is the focus of this research. The background for policy improvement using inaccurate models will now be discussed.

2.2 Policy improvement using inaccurate models

The use of gradient policy iteration is promising in that it guarantees to find a locally optimal policy for some performance measure, $\rho(\pi_\theta)$. However, this is only guaranteed if a perfect representation of the process is known and the performance measure is differentiable by the policy parameters, $\vec{\theta_t}$. The derivatives found are then used to update the policy parameters in the direction of steepest improvement. The computational cost consists of the derivative calculations and policy evaluation. Policy evaluation is just one run per iteration to find the performance metric; however, the more policy parameters, the more derivative calculations are necessary and so this part can make up a large portion of the computation needs. Since only the direction of the derivatives is needed for the update, it is possible to find the direction, $\nabla_{\vec{\theta_t}} f(\vec{\theta_t})$, using a model. Updates from the derivatives, along with policy evaluation from real-life trials can converge in the same way to a locally optimal policy with less real-life trials since the model has taken the workload of the derivative calculations.

In Abbeel et al.(2006) [9], an approach is presented which learns near-optimal policies for a non-stationary MDP described by $M = (S, A, T, H, s_0, R)$, and its inaccurate counterpart defined as $\hat{M} = (S, A, \hat{T}, H, s_0, R)$, where S and A are the state and action sets, respectively, T and $\hat{T}$ denotes the true and model-based process, respectively, H is the number of time steps, and R is the reward function. They explain the algorithmic approach using the following steps:

1. Initialize model for $i = 0$. $\hat{T}^{(0)} = \hat{T}$.
2. Find a locally optimal policy $\pi_{\theta^{(0)}}$ for $\hat{T}$ using a policy search algorithm
3. Evaluate $\pi_{\theta^{(0)}}$ in the real MDP, M, and record the state-action trajectory and the performance, $\rho(\pi_{\theta^{(0)}})$.
4. Construct a new model, $\hat{T}^i$ by adding a time-dependent bias term outside of the control loop, so that the model outputs an estimated real system output from which to calculate derivatives. Specifically set $\hat{f}_t^{(i+1)}(s,a) = \hat{f}_t(s,a) + s_{t+1}^{(i)} - \hat{f}_t(s_t^{(i)}, a_t^{(i)})$ for all times, t.
5. Find the derivative vector $\nabla_{\vec{\theta_t}} \hat{\rho(\pi_\theta)}$ such that the real performance measure $\rho(\pi_{\theta^{(0)}})$ improves from the former iteration.

[3] The step-size, α, must satisfy the stochastic approximation conditions $\sum_{k=1}^{\infty} \alpha_k(a) = \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2(a) < \infty$.

6. Solve for the next policy by a line search in the gradient direction. $\theta^{(\vec{i+1})} = \theta^{(\vec{i})} - \alpha \nabla_{\theta^{(\vec{i})}} \rho(\pi_\theta)$. Use multiple step-sizes, α and choose the best performing policy as policy $\pi_{\theta^{(i+1)}}$.

7. If the line-search does not return an improved policy, then return the current policy and exit. Otherwise, $i \leftarrow i + 1$ and continue at step 3.

The steps are visualized in Fig. 1, which shows the necessary experimental setup, where the *model* plant is represented by the simulation model MDP, $\hat{M}$ and the *real/true* system is represented by the MDP, M. In this paper, M is called the *true* system if it is a simulation or the *real* system if it is a real-life flight. The policy improvement analysis uses the bias from the $\pi_{\theta^{(i)}}$ evaluation flights along with many runs of the model at varying policies to find the partial derivative vector used for policy updating.

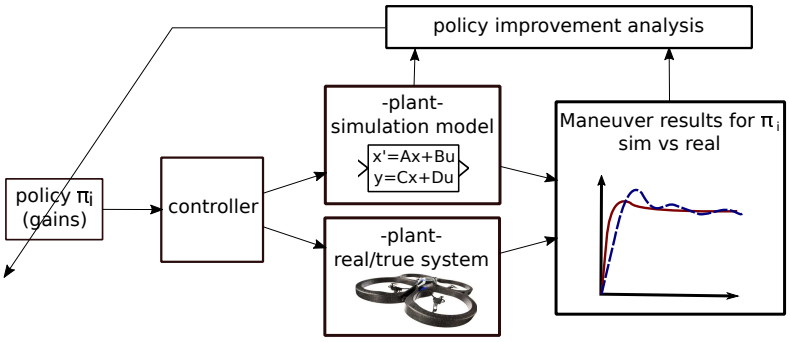

Figure 1: The policy, π_i, is implemented in both the system model and the real-life process. The data from the real-life trial will be used in collaboration with the mathmatical model to find a policy, π_{i+1} which improves the performance of the real system.

The research in this paper takes this concept and expands it to a real-life quadrotor application using a very simple model. The extreme simplicity of the model gives a greater challenge to the approach as it may not capture the necessary derivatives of the real dynamics.

3 Setup

Three experiments are presented in this paper. The first two are fully within simulation and the last is with a real system. In this section, the descriptions of the mathmatical models and true systems are described along with controllers and explanations of the reasoning behind the decisions which were made.

3.1 F-16 in simulation

The first system used as a proof of concept is a model of the F-16 jet fighter. The choice to use this system was based on convenient access to a comprehensive model of the aircraft. From this, a linear time-invariant state space model of the longitudinal dynamics of the F-16 was created with the states: altitude h[ft], pitch angle Θ[rad], velocity V[ft/s], angle of attack α[rad], and pitch rate q[rad/s]. The inputs are thrust F_T[lbs], and elevator deflection δ_{elev}[deg]. The state space equations can be seen in Equation 5.

$$\dot{x} = Ax + Bu \quad \text{where} \quad x = \begin{bmatrix} h & \Theta & V & \alpha & q \end{bmatrix}^T \quad \text{and} \quad u = \begin{bmatrix} F_T & \delta_{elev} \end{bmatrix}^T \tag{5}$$

As described in Section 2, a *true* system and an inaccurate *model* are needed for the experiment. The model will be simulated thousands of times so that the real-life system only needs a few trials. However, a real-life flying F-16 is not so readily available, even for a few trials. Therefore, it was decided that the F-16 state space model would serve as both the true system and the mathematical model. The difference between the 2 are the A and B matrices. The true system is represented by the A,B,C, and D state space matrices which were acquired, and the system model is represented by a perturbed version of the A and B matrices. The perturbations were administered to each element of the matrices as noise with a uniform distribution between $\pm 45\%$. It was discovered through trial and error that a perturbation that is

too small($\pm 20\%$) will end in models that are too similar and therefore will have the same optimal policy and give no challenge to the policy improvement algorithm. If the perturbations are too large (circa $\pm 80\%$) then the model no longer resembles a real physical system and good policies are difficult if not impossible to find for that model (yet good policies are still found for the true system). A perturbation of $\pm 45\%$ was "just right" in that the same set of gains would result in different responses for the two state space sets.

A 6-gain controller was created to control the thrust and elevator deflection. The designated task is to track the pitch angle,Θ, which is set to be a sinusoidal wave with respect to position. The relationship between states, outputs, and gains can be seen in Equations 6, where θ_i represents the policy parameters. The performance metric is the integral absolute error (IAE) between the actual pitch angle and the reference over a 1 minute simulation. Therefore, $\rho(\pi_\theta) = \int |\epsilon| dt$.

$$F_T = \theta_1 V + \theta_2 \tag{6a}$$

$$\delta_{elev} = \theta_3 \Delta\Theta + \theta_4 \alpha + \theta_5 q + \theta_6 \tag{6b}$$

Results for the F-16 experiment are shown in Section 4.4.1.

3.2 Quadrotor in simulation and real flight tests

One experiment in simulation and one on a real-life quadrotor are setup using a simple model of vertical dynamics. Both use the Paparazzi autopilot software (see Section 5.1.3) to control the *true/real system*. The quadrotor simulator built in to Paparazzi uses JSBsim, an open source Flight Dynamics Model, and is used as the *true system* in simulation. The real-life flight experiment uses an AR.drone 2 quadrotor(see Section 5.1.1 as the *real system*. In these experiments, the quadrotor will perform a takeoff maneuver from the ground to an altitude of 3 meters. The details of the simple model created and the controller setup will now be described.

3.2.1 Quadrotor simple vertical model

The simple model of the quadrotor is very crude. The only forces considered are forces of gravity downwards and the upward thrust (which is also the controller output). It is assumed that all rotors give equal thrust and therefore no roll or pitch will develop to tilt the lift vector. The state space matrices for the the plant model are derived from the equations of motion in Equations 7, where F_T is the force of the thrust, m is the mass of the vehicle, and g is gravitational acceleration. z is the axis of altitude, therefore $\ddot{z}$ is vertical acceleration with positive direction up.

$$\sum F = m\ddot{z} = F_T - mg \tag{7a}$$

$$\ddot{z} = F_T \frac{1}{m} - g \tag{7b}$$

$$\begin{pmatrix} \dot{z} \\ \ddot{z} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} z \\ \dot{z} \end{pmatrix} + \begin{bmatrix} 0 & 0 \\ 1/m & -g \end{bmatrix} \begin{pmatrix} F_T \\ 1 \end{pmatrix} \tag{8}$$

In a real flight there will be disturbances to the attitude of the quadrotor. Therefore, the horizontal controller will remain active during the flight tests and only the vertical loop gains will be tuned. The attitude angles should never get too big since the horizontal loop controller will be actively eliminating it. However, there could be repercussions for this omission in the model. For this approach to succeed, the simple quadrotor model only needs to have similar derivatives with respect to the policy parameters, θ, to find the direction of policy improvement for the vertical loop PID gains.

3.2.2 Controller setup

In order for the model to be effective at improving the policy parameters, the design of the controller must be the same for the model simulation as it is for the true system. Otherwise, the gains won't represent the same thing in both controllers and the wrong policy will be learned. In the F-16 experiment, the only difference between the model simulation and the true system was the modified state space matrices. Naturally, these two had the same controller.

In this case, the controller for the true system within the Paparazzi autopilot software is recreated within Simulink. The details of the controller can be found at the Paparazzi webpage [4] [5]

Since the simple model is only used for the vertical loop and assumes no attitude disturbances, there may not be a guarantee that the gains found will be optimal once horizontal movement is needed for other maneuvers. In future work, horizontal gains can be added to the policy parameters for a full controller policy improvement.

3.2.3 Implementation: from simulation to Real-life tests

The benefit of the Paparazzi autopilot is that the same controller software runs with the built-in simulator and the real-life flights. Switching from simulation to an actual flight needs no changes. The learned gains will be different because the simulator is not exactly the same as a real-life quadrotor, however experience shows the simulator to be an accurate predictor of the AR.drone vertical takeoff behavior.

4 Simulation Results

As described in detail in the previous section, two simulation experiments are completed prior to the real-life test. The first simulation learns a 6-gain policy by tracking a sinusoidal flight of an F-16 modeled in Matlab Simulink. The second simulation of a quadrotor is setup in preparation for the real flight. The simple *model* is created in Simulink and the *true* system is represented by JSBsim within the Paparazzi autopilot software. The response from Paparazzi is sent to Matlab where all the policy improvement calculations are completed. In this section, the term "true system trial" is used instead of "real-life trial" and is the same regarding the algorithmic approach. It is just not technically correct to call it a real-life system since it is in simulation. The results for the simulations will now be presented and discussed.

4.1 F-16 simulation results

The results for the F-16 simulation demonstrate the effectiveness of this approach. Ultimately, the policy improvement algorithm works as expected and finds a locally optimal policy for tracking the sinusoidal wave.

The first results to discuss are the results of finding the optimal policy for the *model*. When no decent policy for a real system is known, it is beneficial to begin by finding the optimal policy for the inaccurate model. The initial gains for the model are found by hand to produce a stable, but not optimal, solution. The response of the simulation with the hand-picked gains can be seen in Fig. 2(*right-top*) The model then undergoes a policy search using a Nelder-Mead simplex search algorithm and finds a policy, $\pi_{\theta^{(0)}}$, which optimizes the response for the model simulation. The optimization results are seen in Fig. 2(*left*), where the performance metric, ρ improves over hundreds of iterations of the search. The result, $\pi_{\theta^{(0)}}$ is the best guess for a starting point for a true system trial. The resulting model simulation response is shown to be very good at tracking the reference in Fig. 2(*right-bottom*). Since the model is inaccurate and does not perfectly represent the true system, the same policy, $\pi_{\theta^{(0)}}$, will produce a different, worse, response in the true system. It can be seen that this is true in Fig. 4 in the top plot where policy $\pi_{\theta^{(0)}}$ which was optimal for the inaccurate model, has been used in the true system with a poor tracking performance.

After collecting the data from the first true system trial with policy $\pi_{\theta^{(0)}}$, the policy improvement algorithm can be run to find the derivatives of the inaccurate model and thereafter, the next policy to

[4] Paparazzi Free Autopilot. http://wiki.paparazziuav.org

[5] It is important to note when trying to reproduce these results that Paparazzi is open source and constantly under development. In this experiment, the adaptive nominal throttle function was set to the default value of 0.655, and the takeoff block was modified to fly in vmode = "alt" instead of "climb".

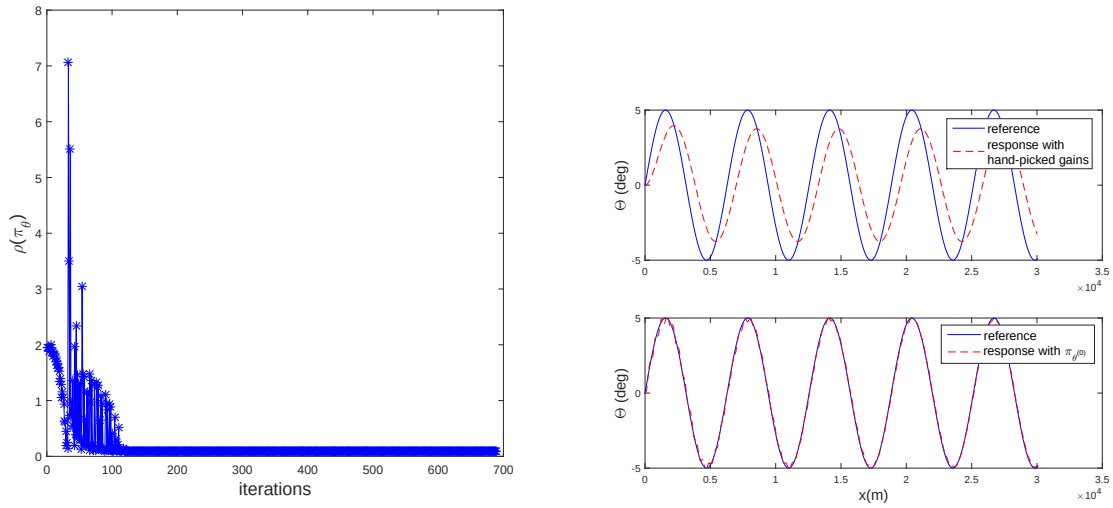

Figure 2: Using the inaccurate model to find a good initial policy. (*left*) As a first step, the optimal policy for the *model* is found using the Nelder-Mead simplex search (non-gradient based) algorithm. The integral absolute error over a 1 minute simulation is used as performance metric and minimized. (*right*) The initial gains (top plot) for the model are found by hand and should be something that is stable but need not be optimal. The policy, $\pi_{\theta^{(0)}}$, from the optimal solution (bottom plot) will be used as a starting point for the true system trials if there is no other known policy for the true system to start from.

try on the true system, $\pi_{\theta^{(1)}}$. This process continues until an optimal has been found. Fig. 3 shows the improvement of the true system flight performance metric after each true system trial. It also shows how the policy has changed over the iterations as well. The first step has the most drastic increase in performance and after that, improvement is achieved in smaller steps. Smaller and smaller changes to the policy parameters are implemented until several runs result in the same error and a local optimal has been reached. The algorithm stops after 27 true system trials, however the parameters don't change significantly after 21 trials and the local near-optimum is reached after only 16 trials. The stopping criteria is strict since it is all run in simulation. In a real-life situation, the trials can be stopped when the user is satisfied with the result or using a custom stopping criteria.

The benefits of this policy improvement can be seen in the results of Fig. 4. Here it is shown that there is a clear improvement between the true system's response with the initial policy, $\pi_{\theta^{(0)}}$, and the final optimized policy, $\pi_{\theta^{(N)}}$. The optimal policy tracks the reference nearly perfectly with a small error at the beginning of the simulation.

4.2 Quadrotor simulation results

The results for the quadrotor simulation policy improvement will now be presented. The simulator used to produce data for the true system trials, is that of JSBsim within the Paparazzi autopilot software. Therefore, it is vital that the Simulink controller which is used for analysis of the derivatives is exactly the same as the controller implemented in Paparazzi. The validation of the simulink controller against the Paparazzi vertical-loop controller will be presented first, followed by the results for the policy improvement based self-tuning gains.

4.2.1 Simulink controller validation against Paparazzi autopilot

Before implementing the policy improvement algorithm, it must first be ensured that the simulink controller (with which the gradient is found) is the same as the Paparazzi controller which controls the real-life trials. The full simulink model of the vertical loop controller and the simple quadrotor model can be seen in Fig 5.

Since we cannot validate the model with the simple quadrotor model inside the closed loop, we must

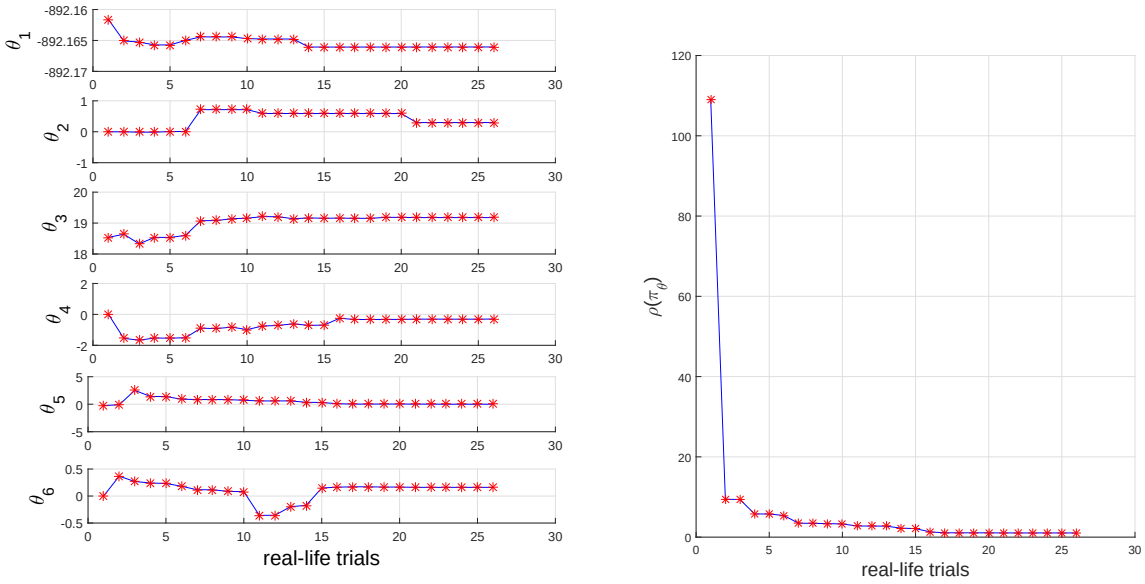

Figure 3: Policy Improvement of the true system system. (*left*) After each true system trial at policy $\pi_{\theta^{(i)}}$, the response is used along with the model to find a new policy, $\pi_{\theta^{(i+1)}}$ based on a local gradient search. The policy consists of the 6 parameters shown. (*right*) The performance metric, $\rho(\pi_\theta)$, is decreased with each true system trial. A local near-optimal is found after 16 trials.

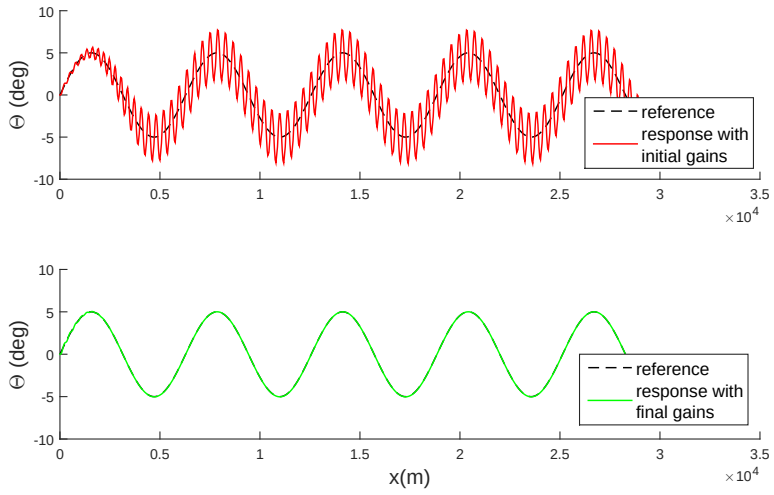

Figure 4: True system responses of the initial policy, $\pi_{\theta^{(0)}}$, and the final policy, $\pi_{\theta^{(N)}}$. The top plot is the response using the model optimal gains, $\pi_{\theta^{(0)}}$. Clearly, the policy that worked for the inaccurate model was not ideal for the true system. After 26 true system trials, a locally optimal policy is found (bottom plot) which tracks the reference Θ very well.

validate the controller in sections. The inputs and feedback signals were taken from a Paparazzi run, so as to confirm that the simulink model will have the same outputs out of the reference model as well as the feedforward and feeback commands. The validation of same outputs can be seen in Fig. 6.

Small disagreements between the Paparazzi and simulink model are due to differences in sample times and round off error. The larger disagreement at the beginning of the feedback command (fb_cmd) plot is due to the summed error integrator not being initialized with the value from Paparazzi. Over time, the

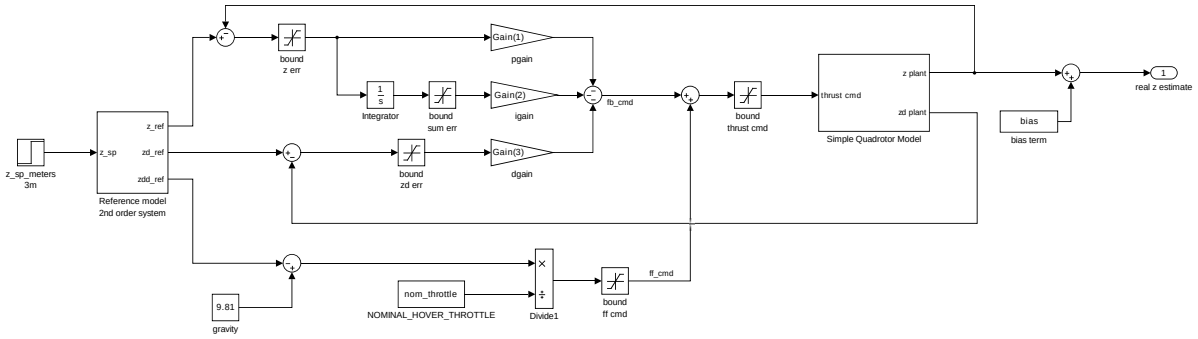

Figure 5: Quadrotor full controller used for policy improvement analysis. It is a replica of the controller from Paparazzi. A bias (determined from previous trials) is added to the final model output to more accurately represent a real-life trial.

plots merge and the discrepancy doesn't effect the performance of the policy improvement algorithm.

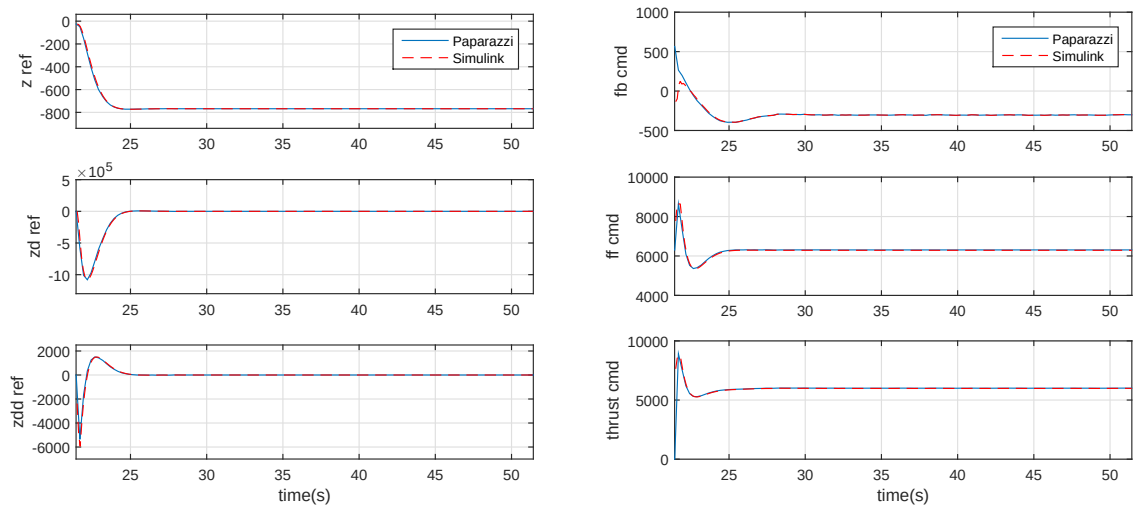

Figure 6: Validation that simulink controller produces the same outputs as Paparazzi. The simulink model was validated using known inputs from a Paparazzi run. The Paparazzi run started with its takeoff command at 21.4 seconds. note: positive z is downward. The units for these plots are from bitwise operations and therefore do not represent a physical value. That the two outputs match is the only important take away.

4.2.2 Policy Improvement

The successful policy improvement to a local optimum is demonstrated in Fig. 7 and the corresponding tables in Table 1. The figures and tables show the results from 2 different initial gain sets. The plot on the left starts with run 1, where the policy is the default gains used within the Paparazzi software for the AR.drone. The take off response is far from optimal as it overshoots its intended altitude and then slowly descends to its setpoint. After each iteration of policy improvement, the gains move toward the locally optimal policy and settle with the policy in run 5. The ending criteria is met when the performance fails twice in a row to improve by greater than 5% from the previously best solution, or if 7 true system/real-life trials have been performed. The plot on the right shows what can be considered a near global optimal. First, the optimal policy solution for the model is found using the Nelder-Mead

simplex search. These gains are used as an initial policy, run 1. The solution found with Nelder-Mead is a very good solution with only 0.04m of overshoot and quick settling time. The error is calculated against the altitude setpoint, however the controller follows the reference trajectory, a 2nd order system. The altitude reference has an error of $0.0913m$ and therefore, we can expect this error to be the best possible performance. The best policy found with our policy improvement algorithm, run 3, results in an error of $0.0940m$.

run #	error(m)	k_p	k_i	k_d
run 1	0.3775	283	13	82
run 2	0.2006	285	44	82
run 3	0.1596	286	102	82
run 4	0.1733	499	177	42
run 5	0.1521	324	116	75

run #	error(m)	k_p	k_i	k_d
run 1	0.1006	1624	1356	195
run 2	0.0988	1644	1365	155
run 3	0.0940	1649	1365	175
run 4	0.0946	1652	1365	195

Table 1: Gains used in each iteration of the policy improvement results. These values correspond to the plots in Fig. 7. (*left*) Initial run uses default Paparazzi gains. (*right*) Initial run uses resulting gains from Nelder-Mead simplex search solution.

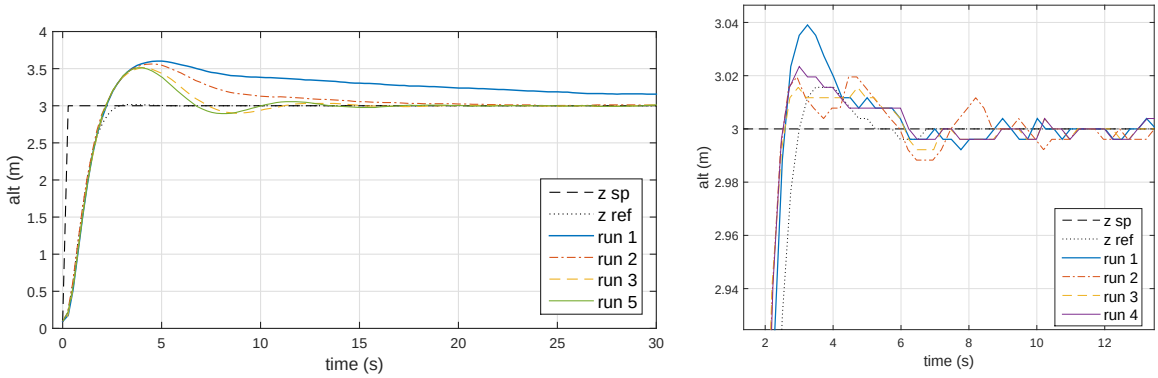

Figure 7: Take off trajectory of true system flights using policy improvement in Paparazzi simulation. The figures correspond to the values from Table 1. (*left*) Initial run uses default Paparazzi gains [283 13 82]. (*right*) Initial run uses resulting gains from a Nelder-Mead simplex search solution [1624 1356 195]. Zoomed-in to overshoot.

5 Quadrotor Flight test

In the flight test, the quadrotor performs a takeoff maneuver from the ground to an altitude of 3 meters. The performance metric to be minimized is the mean absolute error of the altitude against the altitude setpoint. The necessary state information is collected from the Cyber Zoo simulated GPS estimate. Paparazzi, an open source autopilot, is used for the control with the policy parameters represented by the gains of the vertical loop PID controller. Data from the flights is used to run the policy improvement algorithms offline.

The resources used in this process, including the AR.drone 2, the TU Delft Cyber Zoo, and Paparazzi autopilot, will now be described. Thereafter, the results of the flight test experiment are presented.

5.1 Resources

This experiment utilized several resources which will now be described. For a more in depth description of these resources see the paper of a previous experiment [13]. The most up to date information can be found at the websites in the footnotes.

5.1.1 AR.Drone 2.0 quadrotor

Since designing and building a quadrotor is not the main research intention for this experiment, it is very welcome to have a commercial quadrotor available which makes this research not only more convenient for this project but also more accessible to other researchers in the field. A quadrotor is an unmanned aerial vehicle consisting of 4 rotors that spin independently of each other. Adjusting the thrust produced by each rotor gives the necessary control to perform various maneuvers. The Parrot® AR.Drone 2.0 [6], is a commercially available quadrotor intended for recreational use. The AR.Drone 2 is also promising for research and educational purposes [14] as it is relatively inexpensive, and robust to damage. The AR.Drone features a forward facing camera, GPS, WiFi, and precision control. Most importantly for this project is that its built-in software can be overwritten.

Figure 8: The AR.Drone 2.0 is a relatively inexpensive quadrotor MAV that can be purchased for recreational uses or for research. Picture credits by Parrot.com

5.1.2 TU Delft Cyber Zoo

The TU Delft Cyber Zoo [7] was officially opened in March 2014 as a research and test laboratory for ground robots and aerial vehicles. Its space consists of a floor area that is 10 x 10 meters and extends 7 meters high. The space is contained inside an aluminum truss frame which holds nets to ensure the safety of people and surrounding equipment. Also supported on the truss structure are 24 cameras. These cameras are part of the *Optitrack* [8]*: Motive Tracker* optical tracking system. The quadrotor "wears" a special hull with reflective stickers attached so that the cameras can track it. Within the Cyber Zoo boundaries, up to 32 rigid bodies can be tracked with high precision accuracy. The system can also be used as a simulated GPS.

This flight test will use the tracking system for evaluating performance. The complete system as it is used is illustrated in Fig. 9. The computer running the OptiTrack software has a wired connection to the laptop running Paparazzi and from there can broadcast the GPS signal via WiFi to the quadrotor.

5.1.3 Paparazzi Open Source Autopilot

Paparazzi [9] is a fully open source free autopilot for fixed wing and multi-copter UAVs [15]. A version of the software can be acquired freely at GitHub [10].

The quadrotor capabilities in Paparazzi include a model of a standard quadrotor that is close enough to the AR.Drone to function as a good inner loop controller given good tuning and standard non-aggressive maneuvers. Simulation of a flight plan in an intuitive GUI is available in addition to use as a Ground Control Station (GCS) for real flights. An example of the Paparazzi GCS interface can be seen in Fig. 10.

In the indoor setting, Paparazzi connects via WiFi to the quadrotor to give commands and via a wired connection to the Cyber Zoo computer to receive the position of the vehicle. In an outdoor situation, GPS would be picked up by the AR.drone and used there directly.

[6] Parrot AR.Drone 2.0. http://ardrone2.parrot.com/
[7] TU Delft Robotics Institute. http://robotics.tudelft.nl
[8] Optitrack™. http://www.naturalpoint.com
[9] Paparazzi Free Autopilot. http://wiki.paparazziuav.org
[10] Paparazzi GitHub account. https://github.com/paparazzi/paparazzi

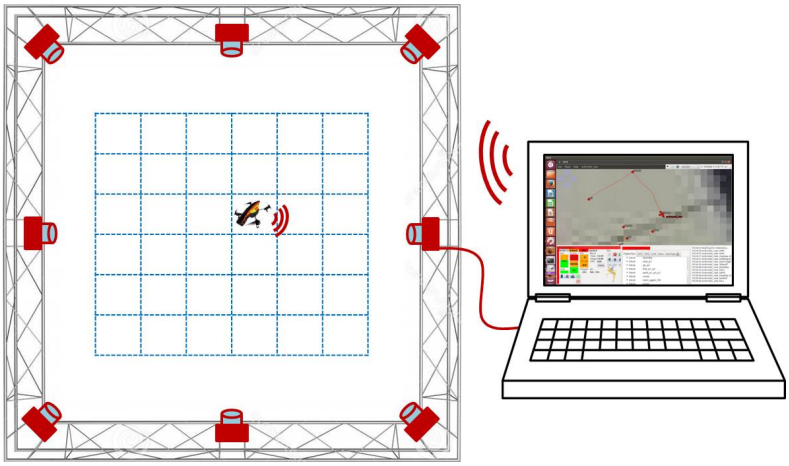

Figure 9: Cyber Zoo experimental setup: 24 cameras track the quadrotor. From this system, the quadrotor receives a simulated GPS signal. That position is transmitted via WiFi between a laptop and the quadrotor. Based on the location state and the memory state, a waypoint command will be sent to the quadrotor for it to execute.

The high level position control of the vehicle is controlled by the flight plan. Waypoints and commands involving these waypoints are written into the flight plan. The autopilot software and the flight plan is compiled and uploaded onto the quadrotor computer. Once uploaded, the names and numbers of the waypoints and types of commands cannot be changed; however, the location of the waypoints can be modified at any time. Therefore, it is important to plan the method for which the waypoints will be used. In this case, only one waypoint was needed and that was the "climb" waypoint which determines the altitude setpoint during takeoff.

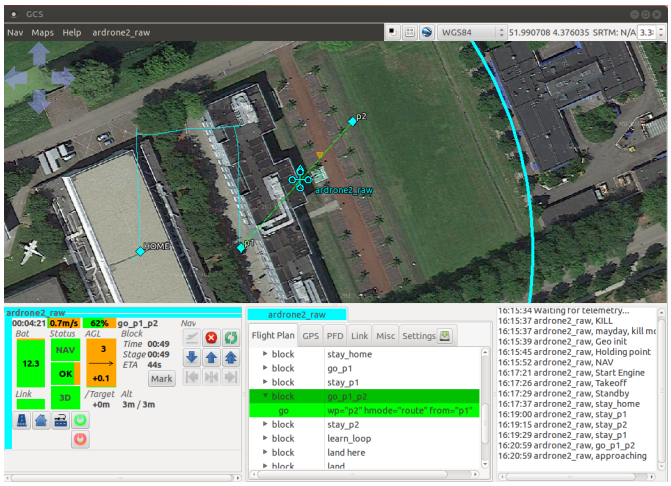

Figure 10: Paparazzi GCS (Ground Control Station) is the main interface for the user. In the GUI it shows a 2D map with a flight path. The quadrotor status is shown in strips box (bottom-left) along with some buttons for common commands. Commands such as "go p1" or "stay p2", can be seen in the Flight Plan tab in the Notebook frame (bottom-middle). Commands that have been executed are printed in the console box (bottom-right).

5.2 System Setup

The controller consists of the Paparazzi autopilot and a flight plan. Once the software is uploaded to the quadrotor computer, the user can send commands from the GCS flight plan via WiFi to the quadrotor.

By calling the block "takeoff", the quadrotor will receive a new altitude setpoint at 3 meters.

5.3 Flight Test Results

The results for the quadrotor policy improvement with real-life flights will now be presented. The structure of the test setup and the results are the same as the quadrotor simulation. Policy improvement with a simple model and real-life trials was performed starting from 2 different initial policies, $\pi_{\theta^{(0)}}$. The first series of flights started from the default gains for the AR.drone in the Paparazzi autopilot software. The second series of flights used an initial policy which was found by the Nelder-Mead simplex search to be the optimal policy for the simple model. The results of these two series are shown in Fig. 11, and the corresponding table, Table 2.

Compare the takeoff trajectory of the default Paparazzi gains, Run 1, in real-life flight (Fig. 11(*left*)) to that of the same flight in simulation (Fig. 7(*left*)) and the same kind of behavior is seen. The quadrotor overshoots its setpoint and then slowly descends towards it. After one iteration of the policy improvement algorithm, a new policy is found. Run 2 performs with about half the error as Run 1 in both the simulation and real-life case. Run 3 shows improvement but not as substantial. Run 4 is not an improvement, which means the step size along the derivative direction was too large. So far, the simulation and the real-life flights show the same progression. At Run 5, they differ. Run 5 is along the same derivative line as Run 4, just not as far. For the simulation, it was an improvement but for the real-life trial it is again not as good as Run 3.

The reason that Run 5 doesn't perform as well as Run 3 could possibly be because Run 3 is already the local optimum, too large of a step size was taken and Run 5's policy is indeed not as good, or external influences have influenced the results. To take a look at the last possibility, an experiment was conducted to see what effects the battery life and variability have on the performance metric. This variability can be inherent in the quadrotor or from external disturbances.

run #	error(m)	k_p	k_i	k_d
run 1	0.5102	283	13	82
run 2	0.2537	286	60	81
run 3	0.2265	301	133	65
run 4	0.2652	301	135	59
run 5	0.2483	301	134	62

run #	error(m)	k_p	k_i	k_d
run 1	0.1218	1624	1356	195
run 2	0.1249	1599	1366	130
run 3	0.1278	1612	1361	163

Table 2: Gains used in each iteration of the policy improvement results. These values correspond to the plots in Fig. 11. (*left*) Initial run uses default Paparazzi gains. (*right*) Initial run uses the resulting gains from a Nelder-Mead simplex search solution.

Multiple takeoff maneuvers were performed successively with the same gains during the life of a single battery on the quadrotor. Tests were performed at the default Paparazzi gains [283 13 82] Fig. 12(*top*) and the Nelder-Mead solution gains [1624 1356 195] Fig. 12(*bottom*). The results show that the battery life does not have an affect on the performance metric. Since the real-life experiment is non-deterministic, it is beneficial in the analysis of the results to know the standard deviation of a takeoff maneuver. The standard deviation of the takeoff performance with the default paparazzi gains is $0.0141m$ and the standard deviation with the Nelder-Mead gains is $0.0066m$. Therefore, it can also be concluded that fine-tuning of the gains on a real-life system is rather difficult with this method given the variability.

While this approach is able to improve the policy from the default paparazzi gains, it fails to improve upon the Nelder-Mead optimized gains. However, in Table 2(*right*) it is shown that the trials after the initial trial are all within the standard deviation of a takeoff maneuver. The initial policy is likely already too good of a performer and therefore, is not able to improve by an amount greater than the variability.

6 Conclusion

Learning through interaction with the environment is one of the key features of reinforcement learning but can be time consuming or dangerous when learning from scratch on a real system. This hybrid approach uses a simple model which takes on much of the computational burden in order to quickly converge to

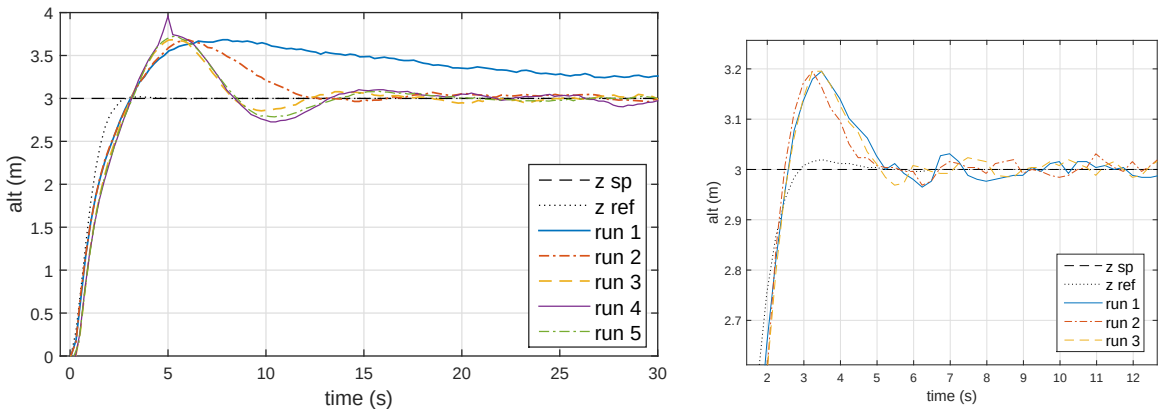

Figure 11: (*left*) Policy improvement starting from initial policy of default Paparazzi gains [283 13 82]. The little peak in Run 4, is not physically possible and is likely a small glitch in the tracking system where perhaps tracking was momentarily lost. (*right*) Policy improvement starting from gains optimized by Nelder-Mead search [1624 1356 195]. Zoomed-in to overshoot.

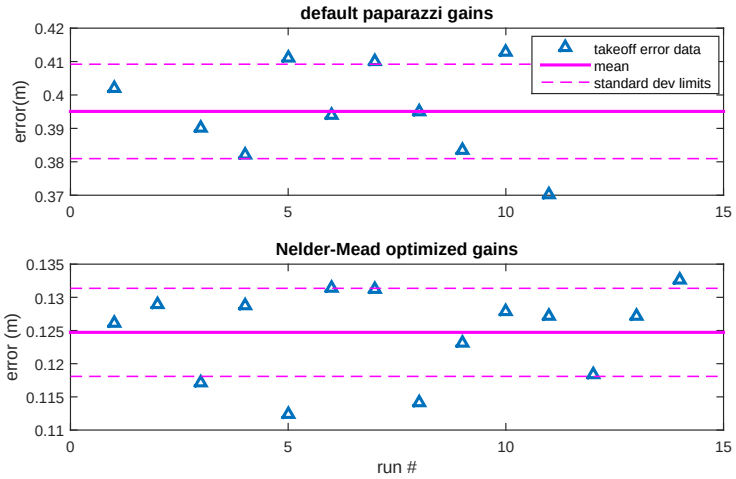

Figure 12: Multiple takeoff maneuvers were performed successively with the same gains during the life of a single battery on the quadrotor. Tests were performed at the default Paparazzi gains [283 13 82] (*top*) and the Nelder-Mead solution gains [1624 1356 195] (*bottom*). The results of this test shows that 1) the battery level does not influence the performance of the takeoff, and 2) the standard deviation of the performance metric depends on which gains are being used and can be used to find the statistical significance of the real-life tests.

a good policy with only a few real-life trials. This method has been shown to be particularly good with policy(gain) improvement of a real-life quadrotor since a local optimum can be found within 3-5 real-life trials and further improvement is only limited by the variability inherent in a real-life application.

This approach has been implemented on an F-16 model in simulation as well as an AR.drone quadrotor in simulation and in real-life.

The conclusions from the F-16 experiment will now be drawn. The results show that the algorithm works as expected. An improved policy with respect to the performance measure is found over each iteration. The largest improvement in performance came between the first and second policy iteration as is expected since the step-size, α, decreases with passing iterations in order to ensure convergence to the local optimal. A near-optimal solution was found after only 16 true system trials.

The quadrotor simulation showed equally promising results. By starting from two different initial gain sets, it was shown that a locally optimal policy could be found. The initial gain set used will highly

effect the performance of the final policy since it is only optimizing locally. The Nelder-Mead simplex search on the simple model provided a starting policy with a much better performance than the default gains from Paparazzi: reducing the initial performance error by 4 times in the real-life trial. Therefore, proving to be a highly effective method for finding the general area of which to start policy improvement algorithm. When starting nearby the global optimum, it is guaranteed that the near-optimal solution will be found. How close it is to the optimum will depend also on the finishing criteria. With stricter criteria, more trials would be performed, but the final solution would become closer to the optimum. In the case of a real-life quadrotor takeoff, the improvement gained by performing more fine-tuning trials would not be important for most applications. Furthermore, the fine-tuning of the gains on a real-life system are shown to be non-beneficial for this maneuver since the performance can be more affected by externally influenced variability than by the gains used. Therefore, the best (or good enough) policy solution was found after 3-5 real-life trials.

References

[1] Richard S Sutton and Andrew G Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[2] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *In Advances in Neural Information Processing Systems 19*. MIT Press, 2007.

[3] Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. Controller design for quadrotor UAVs using reinforcement learning. In *2010 IEEE International Conference on Control Applications*, pages 2130–2135, Yokohama, Japan, September 2010. IEEE.

[4] S. Lupashin, A. Schollig, M. Sherback, and R. D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1642–1648, May 2010.

[5] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Res.*, 4:1107–1149, December 2003.

[6] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(12):81 – 138, 1995.

[7] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12:219–245, 2000.

[8] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[9] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. In *International conference on Machine learning (ICML)*, New York, New York, USA, 2006.

[10] Xin Wang and Thomas G Dietterich. Model-based policy gradient reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 776–783, Washington,DC, 2003.

[11] J.V. Caetano, J. Verboom, C.C. de Visser, G.C.H.E. de Croon, B.D.W. Remes, C. de Wagter, and M. Mulder. Near-hover flapping wing mav aerodynamic modelling - a linear model approach. *International Journal of Micro Air Vehicles*, 5(4), 2013.

[12] Sophie F. Armanini, Coen C. de Visser, and Guido de Croon. Black-box lti modelling of flapping-wing micro aerial vehicle dynamics. In *AIAA Science and Technology Forum: AIAA atmospheric flight mechanics conference*, January 2015.

[13] Jaime Junell, Erik-Jan van Kampen, Coen de Visser, and Qiping Chu. Reinforcement learning applied to a quadrotor guidance law in autonomous flight. In *AIAA Science and Technology Forum: Guidance, Navigation and Control Conference*, pages Paper No. AIAA 2015–1990, 2015.

[14] Tomáš Krajník, Vojtěch Vonásek, Daniel Fišer, and Jan Faigl. AR-Drone as a platform for robotic research and education. In *Research and Education in Robotics - EUROBOT 2011*, volume 161 of *Communications in Computer and Information Science*, pages 172–186. Springer Berlin Heidelberg, 2011.

[15] B.D.W. Remes, D. Hensen, F. Van Tienen, C. De Wagter, E. Van der Horst, and G.C.H.E. De Croon. Paparazzi: How to make a swarm of Parrot AR Drones fly autonomously based on GPS. In *IMAV 2013: Proceedings of the International Micro Air Vehicle Conference and Flight Competition*, Toulouse, France, September 2013.