Deep Learning Quadcopter Control via Risk-Aware Active Learning

Olov Andersson, Mariusz Wzorek and **Patrick Doherty**
{olov.a.andersson, mariusz.wzorek, patrick.doherty}@liu.se
Department of Computer and Information Science
Linköping University, SE-58183
Linköping, Sweden

Abstract

Modern optimization-based approaches to control increasingly allow automatic generation of complex behavior from only a model and an objective. Recent years has seen growing interest in fast solvers to also allow real-time operation on robots, but the computational cost of such trajectory optimization remains prohibitive for many applications. In this paper we examine a novel deep neural network approximation and validate it on a safe navigation problem with a real nano-quadcopter. As the risk of costly failures is a major concern with real robots, we propose a risk-aware resampling technique. Contrary to prior work this active learning approach is easy to use with existing solvers for trajectory optimization, as well as deep learning. We demonstrate the efficacy of the approach on a difficult collision avoidance problem with non-cooperative moving obstacles. Our findings indicate that the resulting neural network approximations are least 50 times faster than the trajectory optimizer while still satisfying the safety requirements. We demonstrate the potential of the approach by implementing a synthesized deep neural network policy on the nano-quadcopter microcontroller.

Introduction

Optimization-based approaches to control and behavior synthesis promise to automatically generate low-level control commands given a model of the dynamics and an objective function. With increasing computational power and maturity of tools, trajectory optimization has been used for increasingly complex behavior like quadcopter flight with slung load, collision avoidance (Geisert and Mansard 2016) or humanoid motion (Tassa, Erez, and Todorov 2012), without the need for tedious custom-tailored controller architectures.

However, in many cases the computational cost of optimization is still too high for the real-time requirements of real robots, particularly on smaller embedded platforms that have limited computational power. Even though there has been a push towards fast solvers (Ferreau et al. 2013; Domahidi et al. 2012; Houska, Ferreau, and Diehl 2011), the domain of problems where real-time operation is practical remains limited.

Policy-based approaches on the other hand are fast to execute but remain a challenge to train. High-dimensional continuous problems are challenging for dynamic programming, and policy-gradient approaches popular in robotics are typically limited to parametric policies with a moderate number of parameters.

It was recently suggested to combine trajectory and policy-based methods, where a particular trajectory optimizer acts as teacher for a deep neural network control policy. Guided policy search (GPS) (Levine and Koltun 2013) alternates between stochastic trajectory optimization under a locally Linear-Quadratic Gaussian (LQG) assumption and fitting a neural network policy via supervised learning. A heuristic cost term or constraint is imposed to make the trajectory not stray too far from the current policy, with the aim of making it easier to learn for the neural network. A similar but deterministic formulation was used by Mordatch and Todorov (2014), which also empirically demonstrated that such a training regime can yield greater sample efficiency.

Learning control policies from an optimizer this way has the advantage that it reduces a difficult policy search problem to supervised learning, a technique that has shown great promise in simulation. In Mordatch et al. (2015) it was used to animate complex movement behaviors and in Zhang et al. (2016) it was used for collision avoidance of static obstacles with a simulated quadcopter. Reports of tests with real hardware are so far few however, where it usually features in safe and controlled environments. In Mordatch et al. (2016) it was combined with an adaptive low-level control scheme for balancing and reaching tasks with a DARWIN-OP, a 45 cm humanoid, and in Levine, Wagener, and Abbeel (2015) vision-based guided policy search was used for a manipulation task on a larger PR-2.

A major concern with real robots is the risk of costly failures. The aim of this paper is to provide a practical approach to construct safe behaviors, generated via offline trajectory-optimization software, into real-time deep policy approximations. The main contributions are that we propose a training procedure with a novel risk-aware resampling step, improving the policy where it matters. This procedure is agnostic of trajectory optimizer, which contrary to GPS is easy to use with existing constrained solvers. We demonstrate that policies maintain safety while being much faster than a state-of-the-art solver for a difficult collision avoidance scenario with non-cooperative moving obstacles. The computational advantages allow us to implement the deep pol-

icy approximation on the microcontroller of a Crazyflie 2.0 nano-quadcopter. To the best of our knowledge this is the first on-board implementation on such a small embedded system, and serves to demonstrate how complex behavior policies can be automatically synthesized and used for computationally constrained platforms.

The remainder of this paper is structured as follows. First we provide the necessary background on trajectory optimization and deep policy approximations, then we introduce the training procedure with risk-aware resampling. The nano-quadcopter platform is briefly covered before we demonstrate the computational gains and safety of the approach in simulation as well as with real flights.

Trajectory Optimization

Consider a robot with the state vector $\mathbf{x} \in \mathbb{R}^n$, control vector $\mathbf{u} \in \mathbb{R}^m$ and uncertain transition dynamics $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$, where $\mathbf{x}$ may include dynamic state like linear and angular velocities. Uncertainty in the dynamics is typically quantified by the fit of some learned function

$$\mathbf{x}_t = \hat{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}; \theta_{\text{dyn}}) + \epsilon_t. \tag{1}$$

We assume discrete-time dynamics, either directly or through some numerical integration scheme like Euler or Runge-Kutta. By solving the discrete-time optimization problem in Eq. (2) at each time step t, we can select the future controls $\mathbf{u}_{t..t+T-1}$ that generate a trajectory $\mathbf{x}_{t+1..t+T}$ with minimal cost $c(\mathbf{x}_{t+1..t+T}, \mathbf{u}_{t..t+T-1})$ from the current state.

$$\begin{aligned} &\underset{\mathbf{u}_t \dots \mathbf{u}_{t+T-1}}{\arg\min} && \mathbb{E}\left[c(\mathbf{x}_{t+1..t+T}, \mathbf{u}_{t..t+T-1})\right] \\ &\text{subject to} && \\ & && \Pr(\mathbf{g}(\mathbf{x}_{t+1..t+T}, \mathbf{u}_{t..t+T-1}) \geq \mathbf{0}) > p. \end{aligned} \tag{2}$$

In many real-world applications, constraint (vector) functions $\mathbf{g}(\mathbf{x}_t, \mathbf{u}_{t-1}) \geq \mathbf{0}$ also need to be satisfied along the trajectory. These can include control saturation, speed limits, or geometric constraints for e.g. collision avoidance. Due to the uncertain nature of the dynamics and state of a real system, the trajectory is stochastic and constraints can only be satisfied with some probability p. This corresponds to a continuous domain Markov decision process with constraints and state uncertainty. The distribution of the state is commonly assumed given by some estimator from observations, $p(\mathbf{x}_t|\mathbf{o}_1 \dots \mathbf{o}_t)$, whose stochastic properties are known and invariant of the state.

Unfortunately, the constrained non-linear probabilistic case is difficult and rarely feasible to solve in anything approaching real-time. For linear-Gaussian problems, uncertainty can be propagated in closed-form using a Kalman filter, or approximated by such. This can also allow easier approximations for some probabilistic constraints (Blackmore and Ono 2009; Vitus and Tomlin 2011). For the special case of unconstrained linear-quadratic Gaussian systems with additive noise, a deterministic solver using the linear-Gaussian mean estimate is also optimal.

Most classical trajectory optimization methods from optimal control instead focus on deterministic optimization problems, where models are deterministic and known, often derived form physical insight. Even when that is not the case, "determinizing" the problem by working with expected values of uncertain variables is often sufficient in practice. Since the problem is discrete-time, it allows a trajectory to be computed with standard optimization packages, where constraints can easily be included. Deterministic constraints will not fully hold for stochastic problems, but one can employ a safety margin. More sophisticated approaches include using Bayesian optimization of deterministic approximations to satisfy probabilistic constraints (Andersson et al. 2016).

Model-predictive control (MPC) is an iterative application of trajectory optimization where at each time step t a trajectory with fixed planning horizon T is computed, typically 10-100 time steps and action $\mathbf{u}_t$ is executed. This makes it robust to inevitable disturbances in real-world applications and controller performance relies on using a planning horizon with sufficient look-ahead for the chosen task. Due to the advantages of optimization-based control, there has been a push towards fast MPC solvers. These are increasingly available, at least for convex problems (Ferreau et al. 2013; Domahidi et al. 2012). They typically have cubic complexity in the number of constraints per time step ($\geq n$), and interior-point solvers can enjoy linear time complexity in the planning horizon (Domahidi et al. 2012). Nonlinear solvers can also be built on these via SQP-like iterations, e.g. Houska, Ferreau, and Diehl (2011). Even with these advances, the computational cost of MPC is often still not practical on real robots, and real-time solutions are reliant on domain simplifications. In the following we instead aim to find robust approximations to trajectory/MPC solvers by training deep neural network policies.

Learning Deep Policy Approximations

Policy-based methods differ from trajectory optimization by instead of at each time step solving for the control signals from the current state, they optimize a parametric policy map from any state to control signals, $\mathbf{u} = \pi_\theta(\mathbf{x})$. If these can be successfully constructed offline, only evaluation is required at each time step. Training policies for the high-dimensional continuous problems in robotics remains a difficult problem, and most success comes from direct policy search methods with a moderate number of parameters (Deisenroth et al. 2013).

However, if we have a near-optimal trajectory optimizer, e.g. an MPC solver, to generate examples $(\mathbf{x}_i, \mathbf{u}_i)$ of near-optimal actions, you can instead reduce this to a simpler problem of learning a supervised learning policy approximation $\hat{\pi}(\mathbf{x}) \approx \pi_{\text{traj.opt.}}(\mathbf{x})$. A difficulty with this is that standard empirical risk minimization assumes i.i.d. data such that $\mathbb{E}_{\mathbf{x},\mathbf{u}}[L(\pi_\theta(\mathbf{x}), \mathbf{u})] = \sum_i [L(\pi_\theta(\mathbf{x}_i), \mathbf{u}_i)]$, but here we have a sequential dependence from the dynamical system $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$.

Disregarding this is suboptimal as errors can accumulate over time, particularly for unstable regions, resulting in poor performance or even dangerous failures. A mitigating circumstance here is that a near-optimal controller has a strong stabilizing effect on the system, such that moderate bounded

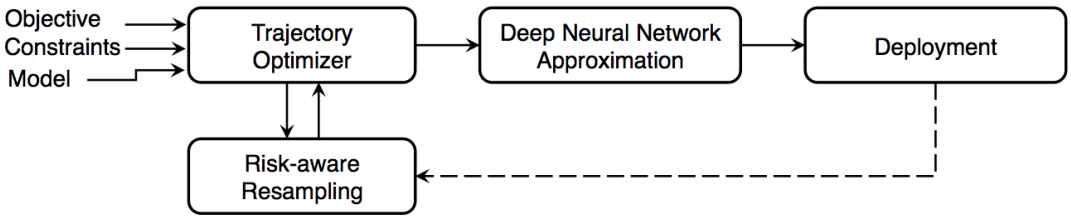


Figure 1: Training procedure for deep neural network policy approximations.

errors in the policy tend to not accumulate, but instead result in more benign steady-state errors. Unfortunately, the error of a supervised policy approximation $\hat{\pi}(\mathbf{x})$ can get arbitrarily bad in regions with little data. Since there is a sequential dependence in the data distribution of $\mathbf{x}_i, \mathbf{u}_i$, this can easily lead to a feedback loop where errors in state and policy approximation accumulate over time.

One popular approach to dealing with this "data mismatch" in learning control policies via supervised learning is DAGGER (Ross, Gordon, and Bagnell 2011), which is formulated as active learning from an oracle. It is a batch algorithm that in each episode fits a policy via supervised learning and then records the encountered state. At the end of the episode it asks the oracle for what the correct action should have been and aggregates the examples.

Another recent approach is guided policy search and similar algorithms, that specifically tightly couple policy learning with a trajectory optimizer (Levine and Koltun 2013; Mordatch and Todorov 2014). Both problems are jointly optimized, coupled by costs or constraints to force the generated, now sub-optimal, trajectory closer to the current policy. Effectively a bias towards trajectories that are easy to learn, with the aim to mitigate accumulating errors. Unfortunately, the coupled nature imposes requirements on trajectories and solver, and has so far only been demonstrated with rather elaborate custom-tailored solvers. Constrained solvers are commonly used in optimization-based control both to encode goals and to keep the robot within a safe operating environment. The iLQR-type (Todorov and Li 2004) trajectory solvers often used for GPS do not work directly with constraints. Emulating constraints in the cost function does not appear to be ideal either due to the recursive Gauss-Newton assumptions inherent to the iLQR backward pass.

Ideally, we would like some method to go from an offline solution using standard trajectory optimization software, to a fast deep neural network approximation that is safe to implement on a real robot. One such approach would be to employ DAGGER. While originally not proposed for deep learning from trajectory optimization, it is conceptually agnostic of both choice of learning algorithm and oracle.

While DAGGER is a simple and powerful technique, it can require many iterations until it converges to a policy that is safe enough for use on a real robot. Especially when a robot has costly failure modes that are relatively rare. In practice, certain states of a domain tend to be more risky than others. E.g. states close to obstacles, high speeds, or large tilt angles for keeping ground robots from tipping over. This highlights a perhaps under-appreciated aspect of learning control policies via a supervised learning transformation. The cost function of the original control problem in Eq. (2), $c(\mathbf{x}_{t+1..t+T}, \mathbf{u}_{t..t+T-1})$, or the constraints, can be very sensitive to errors in $\mathbf{u}_t$ for certain "dangerous" states $\mathbf{x}_t$, while being very forgiving in others. This information is lost when transforming it to an empirical loss $\mathbb{E}_{\mathbf{x},\mathbf{u}}[L(\pi_\theta(\mathbf{x}), \mathbf{u})]$ with i.i.d.data. Since sequential dependence is ignored, we cannot capture the true objective, but we can prioritize accuracy in dangerous regions of the state space.

As we are using a trajectory solver, these might already be encoded as constraints on the solution, in which case the dangerous regions can be trivially defined as some delta around the important constraints.

Risk-Aware Resampling

To combat both the data mismatch problem and heterogeneous costs we propose a simple but effective resampling technique. Formally, assume we can reshape the data distribution to $q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u})p(\mathbf{x}, \mathbf{u})$, where $r(\mathbf{x}, \mathbf{u})$ is a rescaling function assigning higher weight to dangerous regions while still ensuring that $q(\mathbf{x}, \mathbf{u})$ is a proper probability distribution. It is easy to see that this can be formulated as a rescaling of the loss function,

$$\begin{aligned}\mathbb{E}_{q(\mathbf{x},\mathbf{u})}[L(\hat{\pi}_\theta(\mathbf{x}), \mathbf{u})] &= \iint q(\mathbf{x}, \mathbf{u}) L(\hat{\pi}_\theta(\mathbf{x}), \mathbf{u})\, d\mathbf{x} d\mathbf{u} \\ &= \iint r(\mathbf{x}, \mathbf{u}) p(\mathbf{x}, \mathbf{u}) L(\hat{\pi}_\theta(\mathbf{x}), \mathbf{u})\, d\mathbf{x} d\mathbf{u} \\ &= \mathbb{E}_{p(\mathbf{x},\mathbf{u})}[r(\mathbf{x}, \mathbf{u}) L(\hat{\pi}_\theta(\mathbf{x}), \mathbf{u})].\end{aligned} \tag{3}$$

To address both issues we want to select a $r(\mathbf{x}, \mathbf{u})$ that widens the state distribution to make the controller robust to moderate state perturbations from policy approximation errors, as well as assigns more importance to dangerous regions. We used a resampling procedure that first adds a small amount of noise $\mathcal{N}(0, \sigma_r)$ to make the controller robust to moderate state perturbations from policy approximation errors. To amplify dangerous state we do rejection sampling, drawing more samples from dangerous regions by only accepting non-dangerous state with $p = 1/o_r$, where o_r is an odds-multiplier of drawing dangerous state, reflecting a rescaling of the loss. We found that a σ_r of 0.1 standard deviations and o_r of 10 worked well.

This has the dual benefit of both increasing sample coverage and implicitly reweighting the loss function in dangerous regions. Another benefit is that this also allows use of standard supervised learning software without needing to tamper with the loss function.

The full training procedure with this *risk-aware resampling* is seen in Figure 1. Given an objective, constraints and approximate model we can automatically generate wanted robot behavior with a trajectory optimizer using an MPC scheme. For robustness we simulate the system by sampling the stochastic model and state estimator, while recording the chosen state-action $(\mathbf{x}_t, \mathbf{u}_t)$. These then go through the robustness-enhancing risk-aware resampling step based on the domain-specific dangerous regions. Finally, we learn a deep neural network approximation using standard square loss. While we found that this enabled one-shot learning of safe controllers for our application, it is straight-forward to use multiple DAGGER-like iterations.

Deep Neural Network Training

To learn the deep policy approximation we minimize $\mathbb{E}_{\mathbf{x},\mathbf{u}}[L(\pi_\theta(\mathbf{x}), \mathbf{u})]$, where $L(.)$ is simple square loss. The policy is represented by a fully-connected feed-forward deep neural network (DNN) (Bengio, Goodfellow, and Courville 2015), with θ its parameters.

Each DNN layer i is defined by

$$\mathbf{y}_{i+1} = \mathbf{h}_i(\mathbf{W}_i \mathbf{y}_i + \mathbf{b}_i) \tag{4}$$

with network input $\mathbf{y}_1 = \mathbf{x}$, output $\mathbf{y}_N = \hat{\pi}(\mathbf{x})$, and $\mathbf{h}_i(x)$ is the (vector) activation function for layer i. In this paper we use the popular ReLU activation function, which for each neuron j is the scalar function

$$h_{i,j}(x) = \begin{cases} x & \text{if} \quad x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

for all hidden layers i, except the output layer $\mathbf{y}_N$, which is linear. The networks were implemented in Tensorflow[1], a graph-based language for numerical computation, and trained using a consumer Geforce GTX970 GPU. Feed-forward policy evaluation was also implemented on the nano-quadcopter microcontroller, as will be detailed in the experiments section.

Training batches contained about 500 000 examples from the trajectory optimizer, which for our problems took approximately 12 hours to generate. We used the ADAM (Kingma and Ba 2015) stochastic gradient algorithm with mini-batches of size 500 and early stopping. Even just a small amount of dropout (Srivastava et al. 2014), e.g. 1-5%, seemed to be helpful for the larger problems.

Platform

The platform used in the experiments is a Crazyflie 2.0 from Bitcraze[2], seen in Figure 2. The Crazyflie 2.0 is an open-source 7 cm nano-quadcopter platform. It has an empty weight of 27 grams, with 15 grams of payload capacity. The platform is capable of up to 7 minutes of continuous flight. Communication to the ground is realized using the Crazyradio PA (2.4GHz ISM band radio) with 1 km line-of-sight range. On-board integrated sensors include a 10-DOF IMU with accelerometer, gyro, magnetometer and a high precision pressure sensor. The data is collected and processed by a STM32F405 main application MCU (Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash). The Crazyflie uses a complementary filter for roll, pitch and yaw angle estimation and has standard on-board PID controller stabilization for angles given target setpoints.

[1] www.tensorflow.org

[2] www.bitcraze.io

Figure 2: The Bitcraze Crazyflie 2.0 nano-quadcopter.

Experiments

We evaluate the proposed deep policy approximation approach on navigation and collision avoidance tasks with the Crazyflie nano-quadcopter. We first validate the safety of the DNN policies in a difficult simulated collision avoidance scenario and then proceed to implement a suitable policy on the on-board microcontroller.

We follow the training procedure in Figure 1. First we estimate a probabilistic dynamics model of the platform $\hat{f}(.; \theta_{\text{dyn}})$ from Eq. (1). There are a multitude sources of error for a real robotic system, unmodelled dynamics (e.g. turbulence), sensors, latencies and jitter from processing and communication. We therefore do not seek a perfect model and make a linear approximation, a common modeling assumption for quadcopters. We treat the on-board attitude PID loops as part of the dynamics and control the quadcopter using the setpoints as control inputs $\mathbf{u}$. The needed quadcopter state $\mathbf{x}$ is position, velocity and angles. The parameters and errors are identified using domain knowledge and flight data.

Navigation and Collision Avoidance

Efficient collision avoidance outside of controlled environments is in general a hard problem in robotics, particularly for moving obstacles with uncertain motion patterns. Since a trajectory optimizer can take both robot and obstacle dynamics into account, it can produce better behavior. We use a standard objective function with a cost set to reflect distance to the quadcopter navigation goal and small costs on actions, to make flight less aggressive close to the destination. To enforce obstacle avoidance we put constraints on the minimum distance between quadcopter and obstacle positions along the flight trajectory, $\text{dist}(\mathbf{p}_{\text{q,t}}, \mathbf{p}_{\text{o,t}}) \geq 0$. Obstacles with uncertain movement patterns, e.g. humans, are harder since their trajectories are stochastic. We follow

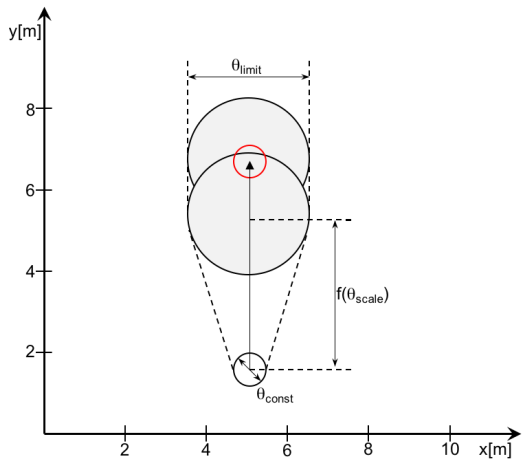


Figure 3: Safety margin for stochastic collision avoidance.

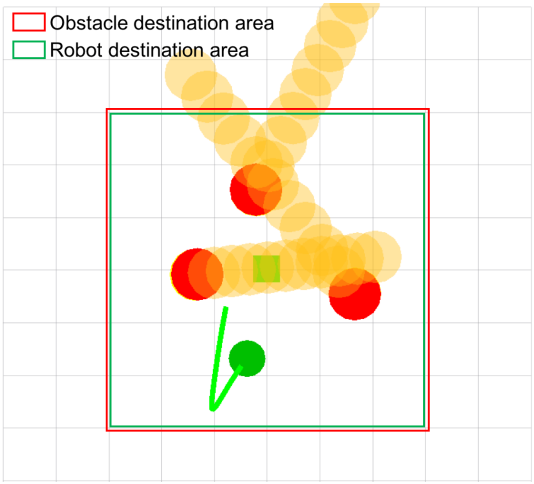


Figure 4: The warehouse scenario with three obstacles.

Andersson et al. (2016) and use safe deterministic approximations with parametric soft margin $m(\theta, \mathbf{x}_t)$, such that $\text{dist}(\mathbf{p}_{\text{q},t}, \mathbf{p}_{\text{o},t}) \geq m(\theta, \mathbf{x}_t)$. We use the parameterization seen in Figure 3, where safe parameter values of $\theta_{\text{limit}} = 1.0$ and $\theta_{\text{scale}} = 0.9$ were found by simulation.

Obstacle constraints are non-convex, and one can use standard non-linear solvers like IPOPT (Wächter and Biegler 2006) since our deep policy learning approach only requires trajectories offline. To offer a better comparison of the possible performance gains of the proposed approach, we instead use the domain-specific iterative MPC solver and scenarios of Andersson et al. (2016), which admitted 10 Hz real-time operation on a desktop CPU.

As benchmark we use the warehouse scenario, seen in Figure 4. This is a difficult scenario where the quadcopter is given navigation goals to pick up green boxes while people move around randomly in the same small area, and without regard for the quadcopter. It is not allowed to fly above humans, so we restrict it to the x-y plane. This makes the collision avoidance task harder, and is a reasonable safety requirement for indoor use. For our proposed risk-aware resampling, we define the high-risk regions to be all state within 1 m of an obstacle. We did not need to tune this.

We consider two problem sizes of this scenario, one moving obstacle and three moving obstacles. We learn DNN policy approximations and evaluate their safety and performance. To perform navigation we need the six-dimensional dynamic state of the quadcopter, and since destination is a free variable we encode it as an input to the policy. Each obstacle is then encoded as another position and velocity pair. We use three hidden layers for each problem size, with the network architectures 10-200-200-200-2 and 18-400-400-400-2 respectively. In terms of computational cost, the largest deep neural network approximation took about 1 ms, over *50 times faster* than the trajectory optimization via MPC. This is illustrated in greater detail by Figure 5.

The safety results of a 20 minute stochastic simulation with one obstacle can be seen in Table 1.

Even in the smaller scenario, a simple supervised DNN approximation using the trajectory optimizer as teacher resulted in possibly costly and dangerous collisions with the obstacle. Looking at the recorded minimum distance, as the obstacles and quadcopter have 0.35 m and 0.07 m radius respectively, the quadcopter managed to go almost straight through it. This is problematic as a small nudge can be acceptable in practice, while collisions at high speed can cause serious damage. This behavior is typical of the data mismatch problem, errors accumulate when the dynamical system runs into regions rarely seen during training.

Scenario	Collisions	Min. Dist.	TTP
Traj. Opt. (MPC)	0	0.14 m	3.12 s
Supervised DNN	9	−0.38 m	3.14 s
DAGGER DNN	**0**	0.21 m	3.14 s
Risk-Aware Resampling DNN	**0**	0.19 m	3.14 s

Table 1: One non-cooperative moving obstacle.

Scenario	Collisions	Min. Dist.	TTP
Traj. Opt. (MPC)	0	0.03 m	4.63 s
Supervised DNN	49	−0.41 m	4.63 s
DAGGER DNN	9	−0.39 m	5.08 s
Risk-Aware Resampling DNN	**0**	0.16 m	5.48 s

Table 2: Three non-cooperative moving obstacles.

Both DNN-augmented DAGGER and the proposed risk-aware resampling can handle this smaller instance. However, as can be seen in Table 2, the larger problem instance is too complicated for robust policies without risk-aware re-

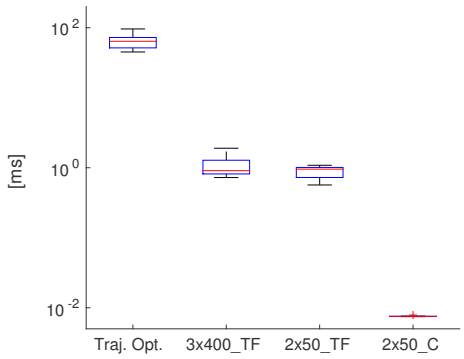


Figure 5: Computational cost of action selection.

sampling, and DAGGER also exhibits the same problematic worst-case performance.

An interesting observation is that there is a trade-off between safety and travel time to package (TTP). The trajectory optimizer oracle , while computationally expensive, can make just the right safety trade-off to quickly pick up the packages. The DNNs are approximations and lose some accuracy. Risk-aware resampling allocates more of their accuracy towards safety, which is precisely the trade-off we wanted to make for operating real robots.

On-board DNN Flights

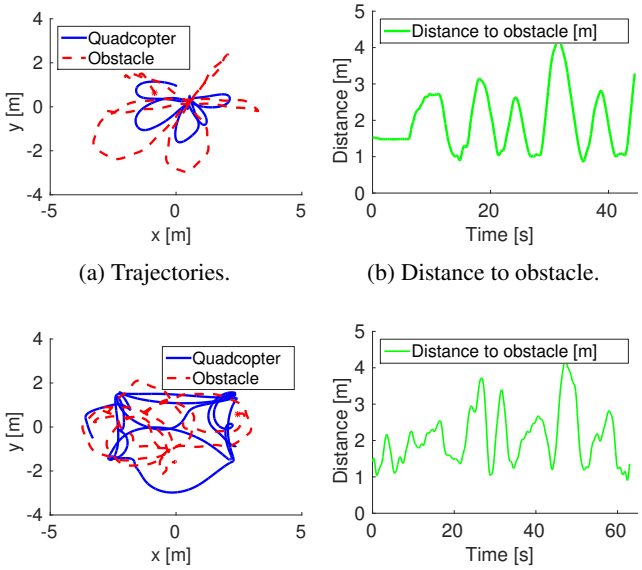
Having ascertained that the method works in simulation, we considered the final step towards a working application, implementing the policy on-board the nano-quadcopter MCU. As generating the behavior via on-board optimization was infeasible, we only evaluated the policy. As the Crazyflie is designed to be small and affordable, it lacks the sensors for positioning and detecting obstacles that may be found on larger more expensive UAVs. We used a room equipped with a motion capture system from Vicon[3] and added a small amount of noise, emulating a laser ranging sensor.

The Bitcraze software running on the MCU uses a real-time operating system kernel from FreeRTOS[4]. Tasks are implemented in the form of modules and scheduled for execution with a fixed update rate. The implementation of our controller was added to the main stabilization task before the target angle set points are set and processed by the inner PID control loops. Each time the external position, velocity and obstacle data are received, the neural network processes them and calculates the target angle setpoints. The sensing updates are sent at 10Hz from the ground.

While processing power was acceptable, the low amount of SRAM turned out to be a bottleneck for large neural networks. Memory is normally not a concern, but this highlights the different computational and memory trade-offs inherent to trajectory and policy approaches. While the original network sizes were chosen for high accuracy in a square-loss sense, as noted this is an imperfect metric of actual

[3] www.vicon.com

[4] www.FreeRTOS.org



(a) Trajectories. (b) Distance to obstacle.

(c) Trajectories. (d) Distance to obstacle.

Figure 6: On-board DNN quadcopter experiments. Top shows it dodging one human obstacle. Bottom shows it flying a rectangular pattern while avoiding one human obstacle.

performance. We found that a 10-50-50-2 architecture could reach sufficient safety for collision avoidance with one obstacle, and we leave real-world tests of larger problem instances for future work.

As Tensorflow is not made for use on resource constrained platforms, we implemented feed-forward operation of the networks in C. As seen in Figure 5, while Tensorflow saw little change in performance due to overhead in the graph execution, the native implementation realized the expected two orders of magnitude speed-up. Another advantage for embedded systems is that the run-time of a DNN policy has low variance, while non-convex trajectory optimization requires complicated iterative numerical algorithms that may take a variable number of steps to converge to a solution. Methods for learning sparser networks or deeper and better abstractions seem like promising directions for leveraging large DNN policies on embedded systems without dedicated hardware.

We share typical flight results[5] in Figure 6. The human obstacle repeatedly walks towards the quadcopter, which in response glides to the side. As can be seen in Figure 6b it maintains a safe distance and never gets closer than 0.8 m. In Figure 6c the quadcopter instead attempts to navigate in a rectangle pattern while the person is walking around randomly. The quadcopter maintains a safe distance throughout and we did not encounter any collisions in our experiments.

Conclusions

Guided by a real quadcopter application, we examined using deep policy approximations to overcome the computational

[5] See supplemental material: youtu.be/xa53w1tyZl0

issues with optimization-based control for robotic platforms. We proposed a novel risk-aware active learning procedure that allows use of both standard trajectory-optimization and deep learning software, while still enabling us to one-shot learn safe policies for difficult quadcopter collision avoidance scenarios. We found the deep neural network policies to be over 50 times faster on the more challenging problems, and the smaller instances even allowed on-board implementation on a nano-quadcopter microcontroller while retaining safety. Deep policy approximations appear to be a promising research direction for embedded platforms that can be expected to improve with advances in deep learning. In particular, methods to learn sparser or deeper and more abstract architectures is an avenue of particular interest, as that could further reduce both memory and computational requirements.

Acknowledgments

This work is partially supported by grants from the Swedish Research Council (VR) Linnaeus Center CADICS, the ELLIIT Excellence Center at Linköping-Lund for Information Technology, the National Graduate School in Computer Science, Sweden (CUGS), the Swedish Foundation for Strategic Research (SSF) project Symbicloud.

References

Andersson, O.; Wzorek, M.; Rudol, P.; and Doherty, P. 2016. Model-predictive control with stochastic collision avoidance using bayesian policy optimization. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 4597–4604.

Bengio, Y.; Goodfellow, I. J.; and Courville, A. 2015. Deep learning. Book in preparation for MIT Press.

Blackmore, L., and Ono, M. 2009. Convex chance constrained predictive control without sampling. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 7–21.

Deisenroth, M. P.; Neumann, G.; Peters, J.; et al. 2013. A survey on policy search for robotics. *Foundations and Trends in Robotics* 2(1-2):1–142.

Domahidi, A.; Zgraggen, A.; Zeilinger, M.; Morari, M.; and Jones, C. 2012. Efficient interior point methods for multistage problems arising in receding horizon control. In *IEEE Conference on Decision and Control (CDC)*, 668 – 674.

Ferreau, H. J.; Kirches, C.; Potschka, A.; Bock, H. G.; and Diehl, M. 2013. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation* 1–37.

Geisert, M., and Mansard, N. 2016. Trajectory generation for quadrotor based systems using numerical optimal control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2958–2964.

Houska, B.; Ferreau, H.; and Diehl, M. 2011. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods* 32(3):298–312.

Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR 2015), San Diego, 2015*.

Levine, S., and Koltun, V. 2013. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 207–215.

Levine, S.; Wagener, N.; and Abbeel, P. 2015. Learning contact-rich manipulation skills with guided policy search. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 156–163.

Mordatch, I., and Todorov, E. 2014. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*.

Mordatch, I.; Lowrey, K.; Andrew, G.; Popovic, Z.; and Todorov, E. V. 2015. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems 28 (NIPS)*. Curran Associates, Inc. 3132–3140.

Mordatch, I.; Mishra, N.; Eppner, C.; and Abbeel, P. 2016. Combining model-based policy search with online model learning for control of physical humanoids. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 242–248.

Ross, S.; Gordon, G. J.; and Bagnell, D. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, 627–635.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Tassa, Y.; Erez, T.; and Todorov, E. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (RSS)*, 4906–4913. IEEE.

Todorov, E., and Li, W. 2004. Iterative linear-quadratic regulator design for nonlinear biological movement systems. In *First International Conference on Informatics in Control, Automation and Robotics*, 222–229 vol. 1. N.P.: INSTICC Press.

Vitus, M. P., and Tomlin, C. 2011. Closed-loop belief space planning for linear, gaussian systems. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2152–2159.

Wächter, A., and Biegler, T. L. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming* 106(1):25–57.

Zhang, T.; Kahn, G.; Levine, S.; and Abbeel, P. 2016. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 528–535.