Model-Based Reinforcement Learning in Continuous Environments Using Real-Time Constrained Optimization

Olov Andersson, Fredrik Heintz and **Patrick Doherty**
{olov.a.andersson, fredrik.heintz, patrick.doherty}@liu.se
Department of Computer and Information Science
Linköping University, SE-58183
Linköping, Sweden

Abstract

Reinforcement learning for robot control tasks in continuous environments is a challenging problem due to the dimensionality of the state and action spaces, time and resource costs for learning with a real robot as well as constraints imposed for its safe operation. In this paper we propose a model-based reinforcement learning approach for continuous environments with constraints. The approach combines model-based reinforcement learning with recent advances in approximate optimal control. This results in a bounded-rationality agent that makes decisions in real-time by efficiently solving a sequence of constrained optimization problems on learned sparse Gaussian process models. Such a combination has several advantages. No high-dimensional policy needs to be computed or stored while the learning problem often reduces to a set of lower-dimensional models of the dynamics. In addition, hard constraints can easily be included and objectives can also be changed in real-time to allow for multiple or dynamic tasks. The efficacy of the approach is demonstrated on both an extended cart pole domain and a challenging quadcopter navigation task using real data.

Introduction

Reinforcement learning holds great promise for robotics. By modeling the robot as a reward maximizing agent that plans ahead one can express a utility function that captures the objective of the task and let the agent figure out the optimal sequence of actions by learning from interactions with the environment.

There are however a number of difficulties with scaling up reinforcement learning methods to real-world robotics problems. The state and action spaces are continuous and often high-dimensional. At the same time operating a real robot is both time and resource intensive, requiring an agent to be able to learn in these environments from a reasonable number of interactions. In addition many tasks in robotics have failure modes that have real costs associated with them. Here the classical approach of considering them terminal states with large negative rewards and letting the agent learn by trial and error is simply not feasible.

To overcome these obstacles a number of improvements have been suggested. To improve sample efficiency there has been renewed interest in model-based reinforcement learning like R-MAX (Brafman and Tennenholtz 2003) where the agent also learns models of the environment. Most of this work is on discrete models, but (Deisenroth, Rasmussen, and Peters 2009) proposes a solution using Gaussian process models. Another line of research is batch RL algorithms like Least Squares Policy Iteration (Lagoudakis and Parr 2003) and Fitted Q-Iteration (Ernst, Geurts, and Wehenkel 2005) that can be bootstrapped with manually collected data. Unless one can find a clever low-dimensional representation of the task all of these still have problems scaling up to higher dimensional problems.

Since learning a policy or value function for high-dimensional continuous domains is challenging, policy search approaches has recently seen more use in continuous robotics problems, e.g. (Bagnell and Schneider 2001; Ng et al. 2004; Peters and Schaal 2006; Kober and Peters 2011). These rely on an expert finding a reduced parametric representation of the policy that is then optimized by evaluating it directly on the robot. Since optimizing policy parameters with a robot in the loop is very time consuming this only works for representations with a small number of parameters or where a simulator is available. Policy search and Gaussian process model-based learning has also been combined in (Deisenroth and Rasmussen 2011) which exploits closed form solutions to the expected reward over time. This allows using policy search with more parameters but requires that the state distribution is well approximated by a Gaussian and imposes restrictions on the reward function.

In this paper we instead explore addressing the original challenges of robotics directly by letting the agent maximize the long term utility of its actions in real-time using learned models of the environment. Trajectory optimization has previously been used in the RL community by e.g. (Abbeel, Coates, and Ng 2010) where demonstrated aerobatic trajectories were tracked using application-specific parametric models. Our work differs from earlier work by learning non-parametric models and considering constrained state spaces which are common in robotics. Sparse Gaussian process models allow the agent to learn from a reasonable number of interactions while being computationally efficient. We sidestep the policy representation problem by iteratively re-planning from the current state in real-time, and we can take

hard constraints into account by solving it as a constrained optimization problem.

The resulting optimization problem is typically studied in non-linear model-predictive control, a type of approximate optimal control. Combining model predictive control with Gaussian process models has also been previously suggested in the control community (Kocijan et al. 2004) where it was used in a small trajectory tracking problem, planning just one step ahead. Since then both fields have advanced considerably and in this paper we show that with recent advances we can now plan hundreds of steps ahead in real-time to feasibly solve challenging reinforcement learning problems with constraints.

We demonstrate the performance and sample efficiency of the approach on an extended cart pole domain and a high-dimensional quadcopter navigation task using using real data. By incorporating constraints it is feasible to produce learning agents that are well-behaved in a target environment and can learn with a minimal of potentially costly errors.

The remainder of this paper is structured as follows. First we formalize the optimization problem, then we introduce the sparse Gaussian process models used to learn the dynamics. Next we introduce the optimization algorithm used with the models to plan an optimal trajectory and finally we demonstrate the efficacy of the approach on simulated and real world data.

Problem Definition

The reinforcement learning problem is informally defined as a rational agent trying to maximize its long-term utility in a potentially unknown environment. In continuous environments this is formalized by a state vector $\mathbf{x} \in \mathbb{R}^p$, a set of actions $\mathbf{a} \in \mathbb{R}^q$, a utility function $U(\mathbf{x}, \mathbf{a})$ and an unknown environment dynamics model $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})$. While this is usually solved by attempting to compute a stochastically optimal policy function over the chosen state representation, we attempt to address the needs of robotics by instead approximating it with the following optimization problem

$$\begin{aligned} \underset{\mathbf{x}_1 \ldots \mathbf{x}_T, \mathbf{a}_1 \ldots \mathbf{a}_{T-1}}{\arg\max} \quad & \sum_{t=1}^{T} U(\mathbf{x}_t, \mathbf{a}_t) \\ \text{subject to} \quad & \mathbf{x}_0 = \text{current agent state}, \\ & \mathbf{x}_t = \mathbb{E}[p(\mathbf{x_t}|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})], \\ & g(\mathbf{x}_t, \mathbf{a}_t) \leq \mathbf{0}, \\ & h(\mathbf{x}_t, \mathbf{a}_t) = \mathbf{0}, \\ & \text{where } t = 1, \ldots, T. \end{aligned} \tag{1}$$

By iteratively solving this finite horizon problem in each time step we get a bounded-rationality agent that plans ahead in real-time to maximize its cumulative utility.

A key advantage of this formulation of the reinforcement learning problem is that we can include hard constraints on the solution, represented by $g(\mathbf{x}_t, \mathbf{a}_t) \leq \mathbf{0}$ and $h(\mathbf{x}_t, \mathbf{a}_t) = \mathbf{0}$, which can be crucial for safe operation of physical robots.

Another advantage is that for many real-world robotics problems the dimensionality of the policy function may be larger than what is feasible to learn without making non-trivial simplifying assumptions on its shape. The environment dynamics $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})$ on the other hand often only depends on a small set of weakly coupled forces acting on the agent. Due to the laws of motion this means that learning a full state transition model decomposes into learning a set of often simple acceleration models and their integration over time.

Unlike Monte Carlo tree search approaches (Kearns, Mansour, and Ng 2002; Walsh, Goschin, and Littman 2010) that have seen increasing popularity for high-dimensional discrete environments we do not attempt to produce stochastically optimal plans, but we make efficient use of gradient information instead. In continuous domains the general constrained stochastic planning problem is intractable. In addition, when controlling a robot in real-time we are more interested in quickly and reliably finding a good solution than finding the best solution, which makes introducing more complexity into the problem unfavorable. A pragmatic heuristic to deal with both observation and model uncertainty is to add a margin of error to the constraints.

Finally, for a policy-based agent to be able to take different objectives into account these need to be represented by variables in the state space to be included in the policy, further increasing the dimensionality of the policy function. In our case it becomes as simple as changing the utility function, which can be done in real-time. As any information the agent has previously learned about the environment is also exploited when performing the new task this allows for a degree of implicit transfer learning between tasks.

Learning the Dynamics

The dynamics model $p(\mathbf{x_t}|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})$ used to plan ahead in (1) is learned from training data by using Gaussian processes, a Bayesian non-parametric regression approach (Rasmussen and Williams 2006).

Gaussian processes

Given a training set of n observations on input variables $\mathbf{X} \in \mathbb{R}^{d \times n}$ and outputs $\mathbf{y} \in \mathbb{R}^d$ where y is corrupted by additive noise $y = f(\mathbf{x}) + \epsilon$, we put a Gaussian process prior on the latent function $f(\mathbf{x})$ and attempt to learn it from data.

A Gaussian process is defined as a set of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and Williams 2006). The process is completely specified by a mean function $m(x)$ and a covariance function $k(x, x')$ that are functions of the input variables. For clarity we assume that all data is standardized with mean zero, turning the covariance function into

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')]. \tag{2}$$

This defines the covariance between input points such that the distribution of any points on $f(\mathbf{x})$ is completely specified by a joint multivariate Gaussian.

Here we use the ARD squared-exponential covariance function which decays with the distance between the points

$$k_f(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left[-\frac{1}{2}\sum_{i=1}^{d}\left(\frac{x_i - x_i'}{\ell_i}\right)^2\right]. \tag{3}$$

The parameters σ_f^2 and ℓ_i represent signal variance and the length-scales respectively which together make up the hyperparameters $\boldsymbol{\theta_f}$. During hyperparameter inference the length-scales will adapt to attenuate the unneeded variables from the problem, termed automatic relevance detection.

The covariance function is used to define matrices of covariances between all the input points $\mathbf{X}$, any set of prediction points $\mathbf{X}_*$ as well as their cross-covariances. We denote these as $\boldsymbol{\Sigma}_{\mathbf{X},\mathbf{X}}$, $\boldsymbol{\Sigma}_{\mathbf{X}_*,\mathbf{X}_*}$ and $\boldsymbol{\Sigma}_{\mathbf{X},\mathbf{X}_*}$ respectively. As points on $p(\mathbf{f}|\mathbf{X},\boldsymbol{\theta}_\mathbf{f})$ are multivariate Gaussian and the noise is Gaussian, the marginal conditional $p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}_\mathbf{f},\sigma_n^2)$ will also be multivariate Gaussian with covariance $\boldsymbol{\Sigma}_{\mathbf{X},\mathbf{X}}+\sigma_n^2 I$.

The predictive distribution $p(\mathbf{y}_*|\mathbf{X}_*,\mathbf{y},\mathbf{X},\boldsymbol{\theta})$ of any new point $\mathbf{x}_*$ therefore follows directly from the conditional distribution of a multivariate Gaussian.

Sparse Approximations

Gaussian processes normally have $O(n^3)$ computational complexity for training, $O(n)$ for computing the predictive mean and $O(m^2)$ for the predictive variance. Since computational performance is crucial for real-time operation we employ a sparse approximation to get this down to $O(m^2 n)$, $O(m)$ and $O(m^2)$ respectively, where m is fixed to some value much smaller than n. We chose a FITC approximation (Snelson and Ghahramani 2006) since it is non-degenerate and does not make the variance estimate overly optimistic (Quinonero-Candela, Rasmussen, and Williams 2007). The FITC approximation adds a set of m latent *inducing* targets and inputs $\{\mathbf{u},\mathbf{X}_\mathbf{u}\}$ where $\mathbf{u}\sim\mathcal{N}(\mathbf{0},\boldsymbol{\Sigma}_{\mathbf{u},\mathbf{u}})$ and $\mathbf{X}_\mathbf{u}$ is typically taken to have a flat prior. By assuming conditional independence between prediction points $\mathbf{f}_*$ and the underlying GP prior so that $p(\mathbf{f}_*|\mathbf{u},\mathbf{f})=p(\mathbf{f}_*|\mathbf{u})$, as well as making any points on $\mathbf{f}$ fully conditionally independent given $\mathbf{u}$, we not only drop the dependence on training data for mean and variance prediction but also get a more manageable complexity for training the hyperparameters. The latent $\mathbf{u}$ are then marginalized out which leads to the following FITC posterior predictive distributions

$$\mathbb{E}[\mathbf{y}_*|\mathbf{X}_*,\mathbf{y},\mathbf{X},\boldsymbol{\theta}] = \mathbf{Q}_{\mathbf{X},\mathbf{X}_*}^T\left[\mathbf{Q}_{\mathbf{X},\mathbf{X}}+\boldsymbol{\Lambda}\right]^{-1}\mathbf{y} \tag{4}$$

$$\begin{aligned}\mathrm{Var}[\mathbf{y}_*|\mathbf{X}_*,\mathbf{y},\mathbf{X},\boldsymbol{\theta}] = {}& \mathbf{Q}_{\mathbf{X}_*,\mathbf{X}_*} \\ & -\mathbf{Q}_{\mathbf{X},\mathbf{X}_*}\left[\mathbf{Q}_{\mathbf{X},\mathbf{X}}+\boldsymbol{\Lambda}\right]^{-1}\mathbf{Q}_{\mathbf{X}_*,\mathbf{X}}\end{aligned} \tag{5}$$

where $\mathbf{Q}_{\mathbf{A},\mathbf{B}} = \boldsymbol{\Sigma}_{\mathbf{A},\mathbf{X}_\mathbf{u}}\boldsymbol{\Sigma}_{\mathbf{X}_\mathbf{u},\mathbf{X}_\mathbf{u}}\boldsymbol{\Sigma}_{\mathbf{X}_\mathbf{u},\mathbf{B}}$ and $\boldsymbol{\Lambda} = \mathrm{diag}[\boldsymbol{\Sigma}_{\mathbf{X},\mathbf{X}}-\mathbf{Q}_{\mathbf{X},\mathbf{X}}+\sigma_n^2\mathbf{I}]$. The joint hyperparameter vector $\boldsymbol{\theta}$ enters the equation through the covariance matrix $\boldsymbol{\Sigma}_{\mathbf{X},\mathbf{X}}$. This is then used in the optimization problem in (1). The marginal FITC likelihood for hyperparameter training similarly becomes

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{X},\theta) = {}& -\frac{1}{2}\mathbf{y}^T\left[\mathbf{Q}_{\mathbf{X},\mathbf{X}}+\boldsymbol{\Lambda}\right]^{-1}\mathbf{y} - \\ & \frac{1}{2}\log|\mathbf{Q}_{\mathbf{X},\mathbf{X}}+\boldsymbol{\Lambda}| - \frac{n}{2}\log 2\pi.\end{aligned} \tag{6}$$

We got good results by simply fixing the inducing inputs $\mathbf{X}_u$ to be a random subset of the data.

Learning in Dynamical Systems

To put this in the context of the reinforcement learning problem in (1), we want to learn the dynamics model $p(\mathbf{x_t}|\mathbf{x}_{t-1},\mathbf{a}_{t-1})$ from the history of state-action transitions $\mathbf{H}$ so that we can compute

$$\mathbb{E}[p(\mathbf{x_t}|\mathbf{x}_{t-1},\mathbf{a}_{t-1},\mathbf{H})]. \tag{7}$$

In this paper we reduce this to learning a model for each dimension separately. For typical robotics problems this can, due to the invariances induced by the laws of motion, be further reduced to learning only a subset of linear and angular accelerations models. Simple Euler integration can then be used to recover the velocity, position, rates and angles of the agent. If state estimates of accelerations are not available they can be recovered by filtering or simple differencing. Each such GP is trained on the history $\mathbf{H}$ of state-action transitions $(\mathbf{x}_t,\mathbf{a}_t)\rightarrow x_{i,t+1}$, which take the place of the GP data terms $\mathbf{X}$ and $\mathbf{y}$ in (4) - (6), for each target state dimension i. We used 10 random restarts of hyperparameter inference to avoid the occasional bad minima.

Constrained Trajectory Optimization

In principle the constrained non-linear optimization problem in (1) can be solved by a standard non-linear programming approach like sequential quadratic programming (Wright and Nocedal 1999). In SQP the non-linear problem is reduced to solving a sequence of convex sub-problems. Since even convex constrained optimization is polynomial in problem size and (1) has $T(p+q)$ variables this may not be feasible in practice.

However, as only the immediately preceding time steps are coupled through the equality constraints induced by the dynamics model, the stage-wise nature of such model-predictive control problems result in a block-diagonal structure in the Karush-Kuhn-Tucker optimality conditions that admit efficient solution. There has recently been several highly optimized convex solvers for such stage-wise problems, on both linear (Wang and Boyd 2010) and linear-time-varying (LTV) (Ferreau et al. 2013; Domahidi et al. 2012) dynamics models.

Since at least the learned dynamics model $\mathbb{E}[p(\mathbf{x_t}|\mathbf{x}_{t-1},\mathbf{a}_{t-1})]$ is non-linear our original problem in (1) falls into the class of non-linear model predictive control problems. With an approach like SQP these can be solved by sequentially applying any convex LTV solver to QP approximations around each point (x_t,a_t) along the current trajectory, resulting in the following structured optimization problem

$$\begin{aligned}\underset{\delta\mathbf{x}_1\ldots\delta\mathbf{x}_T,\delta\mathbf{a}_1\ldots\delta\mathbf{a}_{T-1}}{\arg\max} \quad & \sum_{t=1}^{T}\delta\mathbf{x}_t^T\mathbf{Q}_t\delta\mathbf{x}_t+\mathbf{q}_t\delta\mathbf{x}_t \\ & \qquad +\delta\mathbf{a}_t^T\mathbf{R}_t\delta\mathbf{a}_t+\mathbf{r}_t\delta\mathbf{a}_t \\ \text{subject to} \quad & \mathbf{x}_t=\mathbf{F}_{\mathbf{x},t}\delta\mathbf{x}_{t-1}+\mathbf{F}_{\mathbf{a},t}\delta\mathbf{a}_{t-1}+\mathbf{f}_t, \\ & \mathbf{G}_{\mathbf{x},t}\delta\mathbf{x}_t+\mathbf{G}_{\mathbf{a},t}\delta\mathbf{a}_t+\mathbf{g}_t\leq\mathbf{0}, \\ & \mathbf{H}_{\mathbf{x},t}\delta\mathbf{x}_t+\mathbf{H}_{\mathbf{a},t}\delta\mathbf{a}_t+\mathbf{h}_t=\mathbf{0}, \\ & \text{where } t=1,\ldots,T.\end{aligned} \tag{8}$$

We optimize the relative improvement over a previous trajectory, where $\delta\mathbf{x_t}=\mathbf{x_t^{(k)}}-\mathbf{x_t^{(k-1)}}$, $\delta\mathbf{a_t}=\mathbf{a_t^{(k)}}-\mathbf{a_t^{(k-1)}}$

Algorithm 1 RL-RCO: Real-time Constrained Opt.

For each **episode** repeat:
Learn models on accumulated data with (6).
while not terminal **do**
  if new_objective **then**
    Linearize (7) at $\mathbf{x}_0$ for all time steps.
    Solve (8) until convergence.
  else
    Shift previous trajectory one step forward.
    for i = 1 to num_linearizations **do**
      Linearize (7) around current trajectory.
      Get step direction from (8).
      Update constraint relaxation.
    end for
  end if
end while

and the superscript is the iteration number. The first row of constraints corresponds to linearizations of the model (7) while the second and third are linearizations of the additional state-action constraints from (1). Ideally the objective is a quadratic approximation of the Lagrangian, which includes the utility function as well as Lagrange multipliers to not stray from the region where the constraint linearization is valid. Since we solve it in real-time where the state is noisy and continually changing, a fast answer can be better than asymptotic convergence. For the tasks in this paper we only use quadratic objectives, linear state-action constraints and ignore second order approximations. More elaborate approximations with provable superlinear convergence have been studied in the control community (Diehl, Ferreau, and Haverbeke 2009). These typically put more focus on issues arising from accurately integrating a known physics model. This is less relevant for us since we learn an approximate non-parametric model entirely from data and use simple Euler integration.

As stage-wise convex solver for (8) we chose the recent FORCES (Domahidi et al. 2012) stage-wise LTV MPC solver. FORCES is an interior-point solver with cubic complexity in the number of constraints and linear time complexity in the planning horizon, which is important for long planning horizons typically encountered in reinforcement learning problems.

A high-level view of the algorithm is presented in Algorithm 1. When the agent receives a new objective it plans an initial trajectory through the state-action space by linearizing all stages around the start state $\mathbf{x}_0$. To quickly find an initial trajectory satisfying the constraints in the vicinity of the start state we run the LTV solver once until convergence. For each subsequent step we warm-start with the previous state-control trajectory as the starting point. We optimize for relative improvements using a sequence of linearizations of the learned dynamics at each step of the current trajectory. The warm-starts are crucial as they will iteratively refine the planned trajectory and automatically replan to repair any deviation from it.

We take a soft real-time approach by running a fixed number linearizations per interaction cycle, which in our experiments was set to 4. We found that we did not need to solve the linearized sub-problems until convergence to produce a step with sufficient improvement. For performance constrained applications this is a trade-off that can be tuned. As a bonus multiple small steps might improve global convergence as linearizations are only locally valid, but this needs further study. Strong non-linearities may also introduce bad minima for trajectory optimization but thas has not been a problem so far.

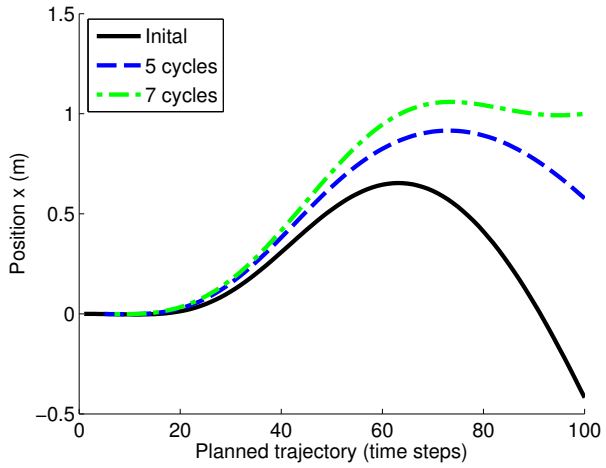

Figure 1: Forward simulation of planned actions for a quadcopter control agent told to reach position $x = 1$. While it plans in a 10-dimensional state-action space, only the target variable is shown.

An example of the iterative improvement of the plan can be seen in Figure 1. The agent was given a new objective and computed an initial trajectory by planning 100 steps ahead using a linearized model. The RL-RCO algorithm runs once per agent interaction cycle and as can be seen it only took 7 cycles for it to converge to a good approximation. It should be noted that the small overshoot seen here is an artifact of the planning horizon and will tend towards zero as it gets closer.

One issue with constrained trajectory optimization is that the optimal trajectory is often along constraint limits. Due to inaccuracies in learned models, solution trajectories, or sensors some constraint boundaries may be temporarily crossed, making the optimization problem unfeasible. At this point we employed a simple constraint relaxation strategy where 5% of the planning horizon was relaxed at a time to make the current solution trajectory feasible.

The RL-RCO agent can also easily be combined with exploration by taking random actions, or even directed exploration by adding a model uncertainty term from (5) to the objective. However, aggressive exploration can make the agent lose control and violate constraints. In our experiments the GP dynamics models were so sample-efficient that we did not need explicit exploration and we therefore leave safe directed exploration as future work.

Experiments

We demonstrate the sample efficiency, accuracy and computational viability of the approach in the experiments below. Using one core of a desktop CPU all agents could run faster than real-time while planning 100 steps into the future. Hard constraints used by RL-RCO were taken directly from the problem description. Terminal ones were tightened by a small margin of error to cope with approximate solutions close to the limit. Dynamics models for the state transition distribution $p(\mathbf{x_t}|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})$ were iteratively learned between episodes for both linear and angular accelerations using sparse Gaussian processes with 40 inducing inputs and ARD kernels. Where noted some initial variable selection was done, but no feature engineering or careful choice of state representation was required.

Extended Cart Pole

Pole balancing tasks are common RL benchmarks and come in many variations. The classic cart pole scenario (Sutton and Barto 1998) consists of a cart with a pole running on a track where the objective is to keep the pole upright and the cart on the track by applying a lateral force to the cart. The state space is $\mathbf{x} = [x, \dot{x}, \theta, \dot{\theta}]$, the action a is a force in the interval -10N to $+10$N and the time step used is 0.02s.

Normally the cart starts in position $x = 0$ with the pole angle $\theta = 0$ (upright). The cart has to stay within $|x| < 2.4$m and $|\theta| < 12$ degrees or the episode is considered a failure and terminates. We extend this by instead of just balancing the pole also have the cart move back to $x = 0$ from a random start position and angle within ± 1m and ± 10 degrees respectively. This is a difficult control problem because the agent may need to move further from the target to tilt the pole forward to then be able to move closer to the goal without violating its constraints. Like with other batch RL methods an initial data collection episode was used to bootstrap the agent, using sinusoid actions capped to 30 interactions. The maximum number of interactions of the RL episodes was then set to 200 and the utility function to the negative square distance to the target position.

Model-free approaches often require at least a couple of hundred episodes for non-trivial versions of the cart pole. A similar positional cart pole using more relaxed constraints was for example solved by NFQ, a fitted Q-iteration approach using neural networks, in 197 episodes (Riedmiller 2005). A related inverted pendulum balancing problem was solved using LSPI in 1000 episodes (Lagoudakis and Parr 2003).

As can be seen from Figure 2 RL-RCO learns in just a few episodes how to successfully move the cart to a target position while balancing the pole. Experiments were run for both fully automatic variable selection (ARD) of the Gaussian process dynamics models as well as with manual selection first using only $[a, \theta]$. Clearly variable selection can help but learning with just ARD took only a few more episodes. In Figure 3 we examine its accuracy by measuring how close it can stay to the target position while balancing the pole. In (Riedmiller 2005) success was defined as staying within 5cm. RL-RCO reaches 1cm accuracy in 5 episodes and 1mm in 20 episodes.

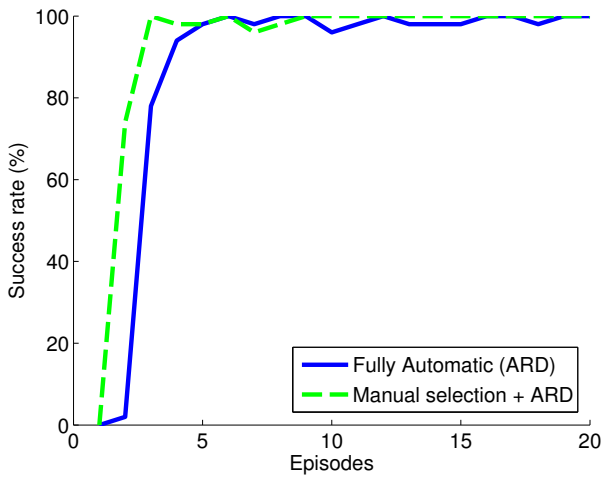

Figure 2: The success rate per episode in terms of the percentage that completed without violating pole or track constraints, averaged over 50 runs.

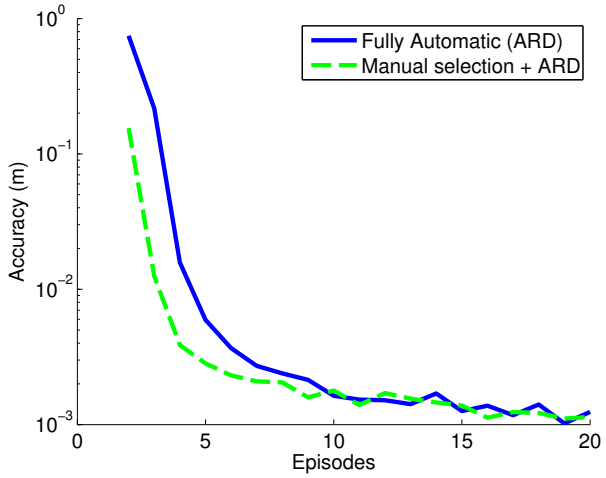

Figure 3: Log plot of the final cart distance to the target position while keeping the pole balanced. The results are averaged over 50 runs.

Great sample efficiency in continuous domains has previously been reported in (Deisenroth, Rasmussen, and Peters 2009) on a 2D inverted pendulum task using GPDP, a dynamic programming approach based on Gaussian processes. Unfortunately, dynamic programming requires extensive pre-computation and tends to scale poorly with dimension. Another sample efficient approach is PILCO (Deisenroth and Rasmussen 2011), a model-based policy search exploiting Gaussian closed form solutions of the expected value integral over the sum of future rewards. Not all reward functions admit closed form solution, making hard constraints non-trivial, and we still need to pre-compute a policy covering the controlled space.

Quadcopter Navigation

We learn positional control for a quadcopter from a batch of real flight data like one might do in a typical robotics application. The resulting controller is then evaluated against a simulator using a previously manually identified physics model.

Initially 8 minutes of training data was collected from manual flight using on-board sensors as well as an external camera-based positioning system, resulting in about 12 000 samples at the controller target time step of 0.04s.

In real robotics tasks the dimension of the state space $\mathbf{x}_t$ might be large. A rigid body representation of a quadcopter is for example 12-dimensional with just positions, velocities, angles and angular rates. However, these are just the integration of accelerations caused by forces acting on the object, which in many cases depend on only a few variables. This makes the model learning problem often much more tractable than trying to learn a policy for the whole state space. We simply decompose the model into learning discrete-time differential models for translational velocities and angular rates. Since terminal failures are unacceptable in a realistic flight scenario we need an agent that is at least flightworthy in one episode using the limited amount of data we have. Using insights from basic physics we exclude unlikely variable combinations from the dynamics models as this was shown to speed up early success rate in the positional cart pole experiment.

To emulate the safety requirements of indoor flight a box constraint of ± 1m/s was set on the velocity components. Using the predictive uncertainty of the GP models, constraints were also manually set on angles, rates and actions to keep the agent within regions where the dynamics were well known.

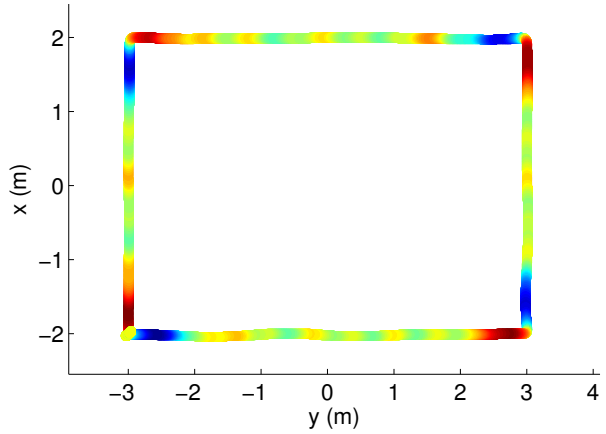

Figure 4: Positional control of quadcopter commanded to fly a rectangle pattern. Blue indicates acceleration, and red deceleration, by getting the quadcopter to tilt in or against the direction of movement. Green is constant velocity.

We used the RL-RCO agent for positional control by defining a utility function that gave reward proportional to the negative square of the distance to a target point. Since RL-RCO plans in real-time it is of course easy to change the objective or constraints if we want it to solve a different task in the same domain. We also used this to fly to waypoints by simply moving the target to the next waypoint when the agent got within 10cm of the previous one. In this experiment we controlled pitch and roll, both rates and angles, as well as velocity and position in the x and y direction. This results in an 8-dimensional state space and 2-dimensional action space. These are the most interesting as a quadcopter first has to tilt to induce lateral acceleration. To maximize long-term reward and avoid large overshoots without violating constraints therefore requires that the controller plans ahead so it is able to start tilting back in time. To the best of our knowledge no other RL approach has solved a high dimensional navigation problem like this one without assuming fixed trajectories or known models in advance.

The results of a simple way-point flight between the points $(\pm 2, \pm 3)$ can be seen in Figure 4[1]. As is evident from the acceleration and deceleration trails it automatically starts tilting back in advance to be able to stop on the targeted location. It should be noted that since the controller was built from real data but the simulator only uses a simplified physics model without drag, we had to fix that input to zero in the lateral acceleration models in order to make a fair comparison.

As can be seen in Figure 5 the controller manages to aggressively pursue its objective within the given velocity constraint of ± 1m/s without overshooting the target position.

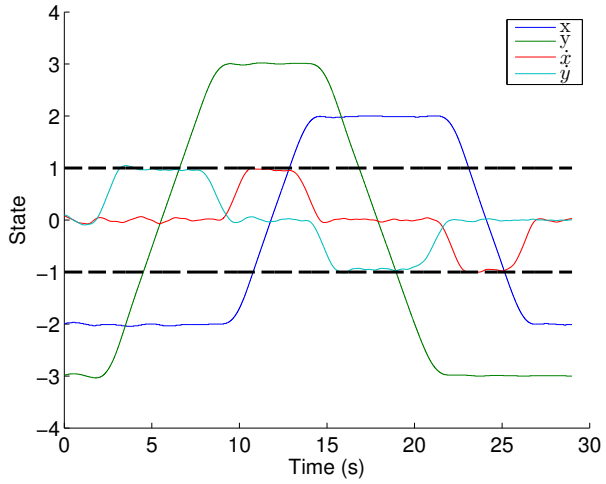

Figure 5: Quadcopter x,y position and velocity while flying the rectangle pattern. The dotted line represents the indoor domain constraint on velocity.

Conclusions

We addressed the challenges of doing continuous reinforcement learning on real robots by letting the agent plan ahead in real-time using constrained optimization instead of precomputing a policy. This has several advantages in that no

[1]Please refer to digital online proceedings for color-coded figures.

high-dimensional policy needs to be computed or stored, hard constraints that are common in robotics can easily be included and the objective can be changed on-the-fly to allow for multiple or dynamic tasks. The benefit of working directly in the continuous domain also allows us to forego issues of discretization and state representation. By combining sparse Gaussian process models with recent efficient stage-wise solvers from approximate optimal control we showed that it is feasible to solve challenging problems in real-time. We demonstrated the performance, sample efficiency and versatility on both an extended cart pole domain and a high-dimensional quadcopter navigation task using real data. To the best of our knowledge no other RL approach has solved such a high dimensional navigation problem without assuming fixed trajectories or known models.

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 Aeronautics Research Council (NFFP6), the Swedish Foundation for Strategic Research (SSF) project CUAS and the Center for Industrial Information Technology CENIIT.

References

Abbeel, P.; Coates, A.; and Ng, A. Y. 2010. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research* 29(13):1608–1639.

Bagnell, J. A., and Schneider, J. G. 2001. Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, 1615–1620. IEEE.

Brafman, R. I., and Tennenholtz, M. 2003. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3:213–231.

Deisenroth, M., and Rasmussen, C. E. 2011. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 465–472.

Deisenroth, M. P.; Rasmussen, C. E.; and Peters, J. 2009. Gaussian process dynamic programming. *Neurocomputing* 72(7):1508–1524.

Diehl, M.; Ferreau, H. J.; and Haverbeke, N. 2009. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In *Nonlinear model predictive control*. Springer. 391–417.

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.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6:503–556.

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.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning* 49(2-3):193–208.

Kober, J., and Peters, J. 2011. Policy search for motor primitives in robotics. *Machine Learning* 84(1-2):171–203.

Kocijan, J.; Murray-Smith, R.; Rasmussen, C. E.; and Girard, A. 2004. Gaussian process model based predictive control. In *American Control Conference, 2004. Proceedings of the 2004*, volume 3, 2214–2219. IEEE.

Lagoudakis, M. G., and Parr, R. 2003. Least-squares policy iteration. *Journal of Machine Learning Research* 4:1107–1149.

Ng, A. Y.; Kim, H. J.; Jordan, M. I.; and Sastry, S. 2004. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16*.

Peters, J., and Schaal, S. 2006. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2219–2225. IEEE.

Quinonero-Candela, J.; Rasmussen, C. E.; and Williams, C. K. 2007. Approximation methods for gaussian process regression. In *Large-scale kernel machines*. Cambridge, MA, USA: MIT Press. 203–223.

Rasmussen, C. E., and Williams, C. K. I. 2006. *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press.

Riedmiller, M. 2005. Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECML 2005)*, volume 3720 of *Lecture Notes in Computer Science*, 317 – 328. Berlin/Heidelberg, Germany: Springer.

Snelson, E., and Ghahramani, Z. 2006. Sparse gaussian processes using pseudo-inputs. In Weiss, Y.; Schölkopf, B.; and Platt, J., eds., *Advances in Neural Information Processing Systems 18*. Cambridge, MA: MIT Press. 1257–1264.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition.

Walsh, T. J.; Goschin, S.; and Littman, M. L. 2010. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings of the 24th Conference on Artificial Intelligence (AAAI)*.

Wang, Y., and Boyd, S. 2010. Fast model predictive control using online optimization. *Control Systems Technology, IEEE Transactions on* 18(2):267–278.

Wright, S., and Nocedal, J. 1999. *Numerical optimization*, volume 2. New York: Springer.