Stochastic Optimal Control on a Real Humanoid Robot

Stochastische Optimale Regelung auf einem Echten Humanoiden Roboter

Studienarbeit von Mike Smyk aus Offenbach am Main

Juli 2016

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Stochastic Optimal Control on a Real Humanoid Robot
Stochastische Optimale Regelung auf einem Echten Humanoiden Roboter

Vorgelegte Studienarbeit von Mike Smyk aus Offenbach am Main

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Elmar Rückert
3. Gutachten:

Tag der Einreichung:

Abstract

For controlling high-dimensional robots, most stochastic optimal control algorithms use approximations of the system dynamics and of the cost function (e.g., using linearizations and Taylor expansions). These approximations are typically only locally correct, which might cause instabilities in the greedy policy updates, lead to oscillations or the algorithms diverge. To overcome these drawbacks, we add a regularization term to the cost function that punishes large policy update steps in the trajectory optimization procedure. We applied this concept to the *Approximate Inference Control* method (AICO), where the resulting algorithm guarantees convergence for uninformative initial solutions without complex hand-tuning of learning rates. We evaluated our new algorithm on two simulated robotic platforms. A robot arm with five joints was used for reaching multiple targets while keeping the roll angle constant. On the humanoid robot Nao, we show how complex skills like reaching and balancing can be inferred from desired center of gravity or end effector coordinates. The results of the Nao are also evaluated on the real robotic platform, whereas most of the stochastic optimal control methods are only evaluated in simulations.

Zusammenfassung

Um Roboter mit vielen Freiheitsgraden zu Regeln benutzen die meisten stochastischen optimale Regelungsalgorithmen Näherungen der System Dynamik und der Kostenfunktion (zum Beispiel mit Hilfe von Linearisierung und Taylor Entwicklung). Allerdings sind diese Näherungen nur lokal korrekt und können zu Instabilitäten in den greedy policy updates, Oszilationen oder Divergenz des Algorithmus führen. Um dies zu Umgehen wird ein Regularisierungsterm eingeführt. Dieser grenzt die Größe der Updateschritte während der Optimierung der Trajektorie ein. Dieses Verfahren wird auf die *Approximate Inference Control* Methode (AICO) angewendet. Der resultierende Algorithmus konvergiert auch für schlechte initiale Lösungen ohne komplexes manuelles anpassen von Lernraten. Der neue Algorithmus wird auf zwei simulierten Roboterplatformen evaluiert. Mit einem Roboterarm mit 5 Freiheitsgraden werden mehrere Ziele angefahren, während der Rollwinkel konstant bleibt. Auf dem humanoiden Roboter Nao werden komplexe Bewegungen gelernt, wie einen Endeffektor Punkt anzufahren während die balance gehalten wird, mit Hilfe eines gewünschten Center of Gravity oder der Endeffektor Koordinaten. Die Ergebnisse der Simulaiton des Nao Roboter werden zusätzlich auf der realen Platform überprüft, wohingegen die meisten stochastischen optimalen Regelungsansätze lediglich in Simulationen überprüft wurden.

Contents

Figures and Tables

List of Figures

1 Contribution

This "Studienarbeit" is part of a journal submission that builds on a robotics conference paper [2]. Therefore, parts of the work in [2] was replicated in this document with the permission of the authors. Section 3, Section 4.1 and Section 5 are fully adopted from [2], as some parts of Section 2. Section 1.1, Section 4.2 and Section 5.1 are new.

1.1 Implementation

During this project a Command Line Interface for the Nao robot was developed, at the beginning of the project present implementation of the Nao SL simulation was fixed, the present implementation of the RSOC was fixed, different cost functions for the RSOC algorithm were implemented and evaluated, it was tried to integrate the SL simulation into the RSOC optimization loop and the trajectories learned by the stochastic optimal controller were evaluated on the real robotic platform Nao. In the following the parts are shortly explained.

1.1.1 Nao Hardware Interface

The first part of the project was to develop an interface to the real humanoid robot Nao. For this, a Command Line Interface (CLI) was developed to execute planned trajectories directly on the robot. In the following the planned trajectories are called movements. The movements have to be available as csv files. The CLI enables to choose different movement sets, execute specific movements, setting the execution time of the movements and in case of a failure reconnect to the robot.
The main challenges were first to understand how to communicate with the robot and then to figure out how the coordinate systems of the joints are arranged, since they are different to the coordinate systems used in SL. Therefor, a script was implemented which prints the values of the joints to the console with 10 Hz.
All the code is provided in the sl_nao Git repository of IAS[1].

1.1.2 Fix Nao SL

When the project started the present version of the SL simulation of the Nao robot was not working, since the IAS version of SL is under active development in contrary to the Nao SL project. The robot was not standing on the ground after spawning. It appeared that there was a major problem with the CMakeLists.txt file of the sl_nao project and some small things in other files, which have been fixed, such that the sl_nao project can now be used within the present version of the IAS implementation of SL again.

1.1.3 Fix Provided Code Base

In the beginning the main idea of this project was to use a given implementation of the RSOC algorithm and optimization loop, develop some cost functions and apply the results on the real robot. Since the present code was not working, the next challenge was to get the present code base to run.
With the fixed code the evaluation of the cost functions and target positions could begin.

1.1.4 Integrate SL Simulation in RSOC

During evaluation of different cost functions it turned out there is a problem with the base coordinate system. Since the goal was to control the robot by getting to a target position with the Center of Gravity (CoG) it is required to have the current CoG in world coordinates. When the a control input is applied to the robot the CoG is moving. A problem occurs if due to the movement also the base coordinate system of the robot moves. The dynamic model of the robot that is used during trajectory optimization does not take the current state of the base coordinate system into account. So in case the base coordinate system is moving during the trajectory the world coordinates of the CoG used in the optimization differ

[1] https://git.ias.informatik.tu-darmstadt.de/

to the real ones when the learned trajectory is applied to the robot. The resulting trajectory will not have the desired behavior of the CoG. Since appending the state of the base coordinate system to the state vector in the optimization loop failed in a previous project an other approach was tried.
To get the base coordinate system moved when applying controls to the robot the current trajectory that was calculated by the optimization algorithm was applied and executed in the SL simulation instead of just using the dynamics model provided by SL. The advantage is on one hand to get a more realistic trajectory that is learned in the end and on the other hand with this approach also falling of the robot can be included in the cost function. Before falling was not modeled by the dynamics model.
To get the SL simulation integrated in the RSOC algorithm a new SL task was implemented, which is able to get trajectories from Matlab using the shared memory library and reset the robot in case it fell down. The new task executes the received trajectory and saves the trajectories of the CoG and the endeffectors. This data is used to calculate the costs of the trajectory.
Two major problems occurred using this approach. The current implementation of SL simulation is much too slow to use it inside the optimization loop. More precisely there is a problem when it is tried to initialize an episode out of Matlab. SL is waiting for a semaphore to be freed, which can take from several seconds up to minutes. Since the simulation would have to be called multiple times in every iteration it would not be practical to use it.
But even if the time of initializing the robot could be shortened there would be an other problem. When using the simulation to evaluate a certain joints state of the robot it is required to get the robot directly to the desired position. In SL it is only possible to apply states the robot should go to from its initial position or its position it is currently in. So when a position is applied to the robot it should go to, the robot can overshoot, destabilize itself and fall even if it would not in this position.
In summary it can be said that using the simulator instead of the dynamics model is not practicable, at least not with SL.

1.1.5 Evaluation on Real Robot Nao

Most stochastic optimal control approaches are only evaluated in simulations, with the exception of e.g. [3] (7-DoF robotic arm) and [1, 4] (7-Dof robotic arm with an attached hand). Hence, as last part of this project the resulting trajectories were evaluated on the real robot Nao, using the CLI mentioned earlier. Even though, there were problems with the base coordinate system it was possible to learn movements without moving the base. The experiments and the results are discussed in more detail in Section 4.2.

2 Related Work

Typical whole body motor control tasks of humanoids, like reaching for objects while walking, avoiding obstacles during motion, or maintaining balance during movement execution, can be characterized as optimization problems with multiple criteria of optimality or objectives. The objectives may be specified in the robot's configuration space (e.g., joint angles, joint velocities and base reference frame), in task space (where objectives such as desired end effector coordinates or center of gravity positions are specified), or in combinations of both. In this paper, we consider control problems in nonlinear systems with multiple objectives in combinations of these spaces.

A common strategy to whole body motor control is to separate the redundant robot's configuration space into a task space and an orthogonal null space. Objectives or optimality criteria of motion are implemented as weights or priorities [5] to the redundant solutions in the null space. While these approaches have been successfully applied to a variety of tasks, including reaching, obstacle avoidance, walking and maintaining stability [6, 7, 8, 9], the application of these methods is typically limited to motor control and can not be directly used for motor planning. It is also unclear how these methods can be applied to motor control problems in nonlinear systems like compliant robots.

Alternatively, in Stochastic Optimal Control (SOC) problems [10], a movement policy is optimized with respect to a cost function, which combines the different criteria of optimality with different weightings. For nonlinear systems, SOC methods use approximations of the system dynamics and of the cost functions, e.g., through linearizations and 2nd order Taylor expansions. These approximations are only locally correct and the updates of the policy may become unstable if the minima is not close to the points of the linearizations, or may oscillate in the case of multiple solutions.

Many SOC methods address this issue and implement regularizations on the *algorithmic level*. E.g., in the *iLQG* method [11] a diagonal regularization term is added to the control cost Hessian[1], and in an extension [12], it was suggested to penalize deviations from the state trajectory used for linearization rather than controls. In [12] this approach is shown to work for a 22-DoF humanoid robot in simulation to perform complex tasks like getting up from ground. In [3] *iLQG* is used to learn a compliant optimal feedback controller for a real 7-DoF robot arm to solve reaching tasks. A drawback of this approach is that the additive regularization term needs rapid re-scaling to prevent divergence and accurate fine-tuning of a learning rate to find good solutions, which is challenging and increases the computational time of the algorithm.

Probabilistic planning methods that translate the SOC problem into an inference problem, typically implement learning rates in their belief updates [13] or in the feedback controller [14]. In [13] the approximate inference control (AICO) is used to control a 39-DoF humanoid robot in simulation performing several reaching tasks. In [1] a real 14-DoF Schunk arm is controlled to perform a real world blocks world scenario, by using AICO. However, in nonlinear systems, both strategies are suboptimal in the sense that even with a small learning rate on the beliefs the corresponding control updates might be large (and vice-versa, respectively).

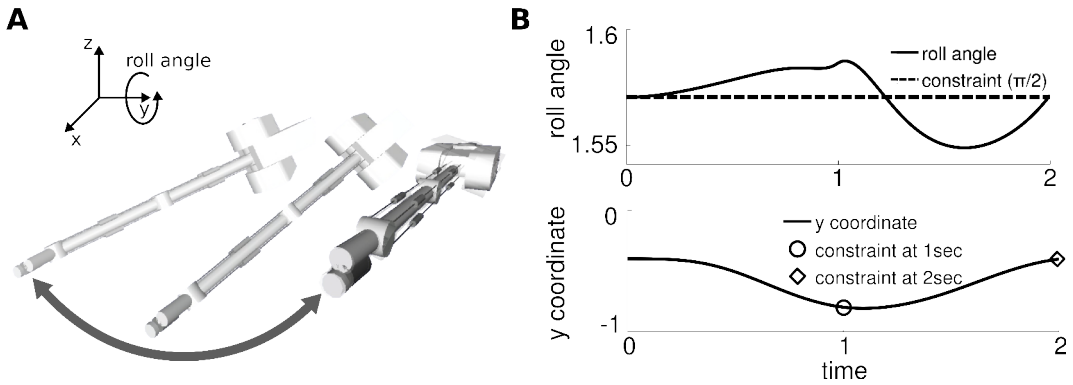


Figure 2.1: A 5-degree-of-freedom robot arm has to reach for a via-point (the posture on the left in A) and return to its initial pose (the posture on the right in A). The reaching task is encoded in four task objectives, i.e., three Cartesian coordinates and the roll angle of the end effector. The inferred trajectories for the y coordinate and the roll angle, including the objectives, are shown in (B).

[1] The update step in the trajectory optimizer corresponds to a Gauss-Newton Hessian approximation [12].

Instead of using one linearization of the system the SOC problem can be linearized in each iteration around a current minimal cost state [15]. In contrast to the AICO approach there is no need of a additional line search to ensure convergence. For systems with imperfect sensing this approach can also be extended to work in the robots belief space [16]. A drawback of those approaches is the requirement of the inverse stochastic dynamics.
All of the introduced approaches depend on the quality of the used model and on the linearizations. A model-free approach for learning a controller is Pi^2 introduced in [17]. It is a reinforcement learning approach that is derived from SOC methods. Pi^2 is used to learn full-body motor skills on a 34-DOF humanoid robot [17] and to perform reaching and pick-and-place tasks for a real 7-Dof robotic arm with a three fingered 4-DoF hand [4]. However, applying reinforcement learning to real robotic platforms is time-consuming and costly. The learning phase can be done in simulation, but then the approach again depends on the simulation model of the robot and the benefit of the reinforcement learning approach, being model-free, is lost.

2.1 Overview Table

Method	Experiments	Notes	Diff to RSOC	Year	Ref.
RSOC	• 5-DoF robot arm (sim) • Nao (sim and real)	• Forward and backward propagation (AICO) • Uses AICO and applies regularized cost function to it • Computes a controller		2014	[2]
iLQG	• Inverted Pendulum (sim) • Model of human arm (sim) • 22-DoF humanoid model (sim)	• Locally-optimal feedback control of constrained nonlinear stochastic systems • No noise compensation • Dynamic programming method that use quadratic approximations to the optimal cost-to-go function • Only for fully observable system	regularization on algorithmic level [“diagonal regularization term is added to the control cost Hessian”]	2005 2012	[11] [12]
iLQG	• 7-DoF robot arm (real)	• Applied iLQG to real rootic platfrom to achive a compliant optimal feedback controler • Had to specify real robot contraints like effects of coriaolis and centripedal effects, joint friction and gravity loading depending on joint angles	For Nao the effects of the forces (e.g. coriolis) can be ignored, since torques on joints occur from those are small. For barrett more important.	2010	[3]
AICO	• 39-DoF humanoid robot (sim)	• Local approximate solution to the SOC problem that is fast to compute • Generlaize algorithms like iLQG to non-LQG problems	Regularization on algorithmic level	2009	[13]

ELQR	• iRobot Create differential-drive robot (sim) • Quadrotor helicopter (sim)	• Applies to non-Linear dynamics and non-quadratic cost functions • Linearizes dynamics and quadratizes cost function in each iteration around current min-cost state • Uses ”LQR-smoothing” to search for state which produces minimal total cost • No line search needed	No line search (AICO); Uses “LQR-smoother”; Not used for humanoid robots	2013	[15]
SELQR	• Car-like Robot in a 2-D Environment (sim) • Quadrotor in a 3-D Environment (sim) • Medical Needle Steering for Liver Biopsy (sim)	• Can optimize in state space and in belief space (if states are not observable or hard to observe because of noise) • “LQR-smmothing” based on Kalman smoother • Extension to POMDPs (to plan in belief space)	Not used for humanoid robots	2015	[16]
Pi^2	• 34-DOF Robot (sim)	• Model free reinforcement learning • Only forward roll-outs • Computes a DMP	No regularizatoin of roll-outs; Concentrates on manipulation tasks; Drawback model-free: Necessary to interact with robot in every iteration	2010 2012	[17] [4]

3 Stochastic Optimal Control (SOC) Methods

We consider finite horizon Markov decision problems[1]. Let $\boldsymbol{q}_t \in \mathbb{Q}$ denote the current robot's state in configuration space (e.g., a concatenation of joint angles, joint velocities and reference coordinates in floating base systems) and let vector $\boldsymbol{x}_t \in \mathbb{X}$ denote task space features like end effector positions or the center of gravity of a humanoid (these features will be used to specify a cost function later). At time t, the robot executes the action $\boldsymbol{u}_t \in \mathbb{U}$ according to the movement policy $\pi(\boldsymbol{u}_t|\boldsymbol{q}_t)$.
The chosen action at the current state is evaluated by the cost function $C_t(\boldsymbol{q}_t, \boldsymbol{u}_t) \in \mathbb{R}^1$ and results in a state transition characterized by the probability $P(\boldsymbol{q}_{t+1}|\boldsymbol{q}_t, \boldsymbol{u}_t)$. In Stochastic Optimal Control (SOC), the goal is to find a stochastic policy π^* that minimizes the expected cost

$$\pi^* = \underset{\pi}{\operatorname{argmin}} \langle C_T(\boldsymbol{q}_T) + \sum_{t=0}^{T-1} C_t(\boldsymbol{q}_t, \boldsymbol{u}_t) \rangle_{q_\pi} \ , \tag{3.1}$$

where the expectation, denoted by the symbols $\langle \cdot \rangle$, is taken with respect to the trajectory distribution

$$q_\pi(\boldsymbol{q}_{0:T}, \boldsymbol{u}_{0:T-1}) = P(\boldsymbol{q}_0) \prod_{t=0}^{T-1} \pi(\boldsymbol{u}_t|\boldsymbol{q}_t) P(\boldsymbol{q}_{t+1}|\boldsymbol{q}_t, \boldsymbol{u}_t) \ , \tag{3.2}$$

where $P(\boldsymbol{q}_0)$ is the initial state distribution.

3.1 Bayesian inference for control

An interesting class of algorithms to SOC problems have been derived by reformulating the original Bellman formulation in (3.1) as an Bayesian inference problem [20, 21, 22, 23]. Instead of minimizing costs, the idea is to maximize the probability of receiving a reward event ($r_t = 1$) at every time step

$$p(r_t = 1|\boldsymbol{q}_t, \boldsymbol{u}_t) \propto \exp\{-C_t(\boldsymbol{q}_t, \boldsymbol{u}_t)\} \ . \tag{3.3}$$

Note that the idea of turning the cost function in Eq. (3.1) into a reward signal was also used in operational space control approaches [24, 25].
In the probabilistic framework, we want to compute the posterior over state and control sequences, conditioning on observing a reward at every time step,

$$p_\pi(\boldsymbol{q}_{0:T}, \boldsymbol{u}_{0:T-1}|r_{0:T} = 1) = \exp\{-C_T(\boldsymbol{q}_T)\} q_\pi(\boldsymbol{q}_{0:T}, \boldsymbol{u}_{0:T-1}) \prod_{t=1}^{T-1} p(r_t = 1|\boldsymbol{q}_t, \boldsymbol{u}_t) \ . \tag{3.4}$$

For efficient implementations of this inference problem, a number of algorithms have been proposed that apply iterative policy updates assuming that all probability distributions can be modeled by an instance of the family of *exponential* distributions [13, 26, 27]. We will restrict our discussion on the Approximate Inference Control (AICO) algorithm with Gaussians [13].

3.2 Approximate inference control with Gaussians (AICO)

We consider system dynamics of the form $\boldsymbol{q}_{t+1} = f(\boldsymbol{q}_t, \boldsymbol{u}_t) + \epsilon$ with ϵ denoting zero mean Gaussian noise. In AICO (with Gaussians), the system dynamics are linearized through 1st order Taylor expansions, i.e., $P(\boldsymbol{q}_{t+1}|\boldsymbol{q}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{q}_{t+1}|A_t\boldsymbol{q}_t + a_t + B_t\boldsymbol{u}_t, Q_t)$, where the state transition matrix A_t, the linear drift term a_t and the control matrix B_t are often computed with derivatives simulated through finite differences methods. The numerical stability of AICO also depends on the accuracy of the linearized model, we will therefore additionally compare to an approximation of the system dynamics,

[1] Note that the same principle of regulating the update steps in trajectory optimization can also be applied to planning algorithms in infinite horizon problems such as [18, 19]

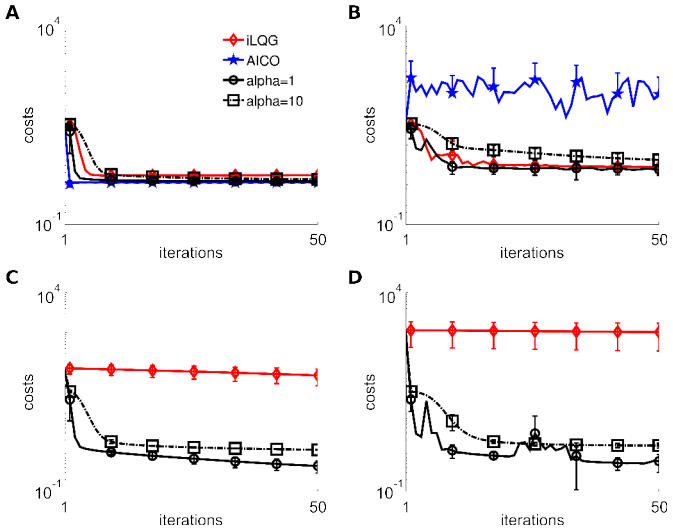


Figure 3.1: Comparison of the convergence properties of *iLQG*, AICO and our robust variant, where the rate of convergence is controlled via the parameter α. In the top row (A-B), the model of the forward dynamics was approximated by a *pseudo dynamics* model [1]. In the bottom row, an analytic forward dynamics model of a 5-degree-of-freedom robot arm was used. The panels in the first column denote the costs of the planning algorithms applied to a simple task, where the robot arm has to reach for an end effector target and return to the initial state. In the second column (B,D), the robot has to keep additionally the roll angle constant (at $\pi/2$). Shown are the mean and the standard deviations for 10 initial states '$\boldsymbol{q}_0$ sampled from a Gaussian with zero mean and a standard deviation of 0.05.

where controls $\boldsymbol{u}_t$ correspond directly to joint accelerations[2]. We will refer to this approximation as *pseudo-dynamic* model.

We propose to add a regularization term to the cost function. Before explaining the regularization term in more detail, we briefly discuss how different objectives are implemented in AICO. In the simplest case, the task-likelihood function in (3.3) can be split into separate state and a control dependent terms, i.e.,

$$p(r_t = 1|\boldsymbol{q}_t, \boldsymbol{u}_t) = \mathcal{N}[\boldsymbol{q}_t|\boldsymbol{r}_t, R_t]\,\mathcal{N}[\boldsymbol{u}_t|\boldsymbol{h}_t, H_t] \ , \tag{3.5}$$

where, for analytical reasons, the Gaussians are given in *canonical* form, i.e., $\mathcal{N}[\boldsymbol{u}_t|\boldsymbol{h}_t, H_t] \propto \exp(-1/2\,\boldsymbol{u}_t^T H_t \boldsymbol{u}_t + \boldsymbol{u}_t^T \boldsymbol{h}_t)$. Note that the vector $\boldsymbol{r}_t$ in (3.5) denotes the linear term for the Gaussian distribution and must not be confused with the auxiliary variable $r_t = 1$ in (3.3) denoting a reward event. By inserting (3.5) in (3.3) we obtain the quadratic costs,

$$C_t(\boldsymbol{q}_t, \boldsymbol{u}_t) = \boldsymbol{q}_t^T R_t \boldsymbol{q}_t - 2\boldsymbol{r}_t^T \boldsymbol{q}_t + \boldsymbol{u}_t^T H \boldsymbol{u}_t - 2\boldsymbol{h}_t^T \boldsymbol{u}_t \ . \tag{3.6}$$

The state dependent costs, encoded by $\mathcal{N}[\boldsymbol{q}_t|\boldsymbol{r}_t, R_t]$, can be defined in configuration space[3], in task space[4], or even in combinations of both spaces [22].

On the algorithmic level, AICO combines forward messages and backward messages to compute the belief over trajectories. We represent these Gaussian forward message by $N[\boldsymbol{q}_t|\boldsymbol{s}_t, S_t]$, the backward message by $N[\boldsymbol{q}_t|\boldsymbol{v}_t, V_t]$, and the belief by $N[\boldsymbol{q}_t|\boldsymbol{b}_t, B_t]$. The recursive update equations are given in [13] and in [14] where an implementation which additionally implements control constraints (otherwise $\boldsymbol{h}_t = 0$) is given.

We can also compute the most likely action given the task constraints. By doing so, in the case of AICO with Gaussians, we obtain a time varying linear feedback controller

$$\boldsymbol{u}_t^{[n]} = \boldsymbol{o}_t + O_t \boldsymbol{q}_t \ , \tag{3.7}$$

where $\boldsymbol{o}_t$ is an open loop gain and O_t denotes the feedback gain matrix (n denotes the iteration).

---

2 For a single joint with $\boldsymbol{q} = [q, \dot{q}]^T$, the matrix $A = \left(\begin{smallmatrix} 1 & \tau \\ 0 & 1 \end{smallmatrix}\right)$, $a = \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)$, and $B = \left(\begin{smallmatrix} \tau^2 \\ \tau \end{smallmatrix}\right)$, where τ denotes the time step.

3 Reaching a goal state $\boldsymbol{g}^* \in \mathbb{Q}$ in configuration space can be encoded by $\boldsymbol{r}_t = R_t \boldsymbol{g}^*$ where the precision matrix R_t scales the importance of different dimensions.

4 Let $\boldsymbol{x}^* \in \mathbb{X}$ denote a desired end effector position and let $\boldsymbol{x} = f(\boldsymbol{q})$ be the forward kinematics mapping and $J(\boldsymbol{q}_t) = \partial f / \partial \boldsymbol{q} | \boldsymbol{q} = \boldsymbol{q}_t$ its Jacobian. We can now obtain a Gaussian task likelihood by approximating the forward kinematics by its linearization through the Jacobian, i.e., $\boldsymbol{x} \approx f(\boldsymbol{q}_0) + J(\boldsymbol{q} - \boldsymbol{q}_0)$. The parameters of the Gaussian are then given by $\boldsymbol{r}_t = J^T C\big(f(\boldsymbol{q}_0) - \boldsymbol{x}^*\big)$ and $R_t = J^T C J$, where the diagonal elements of the matrix C specify the desired precision in task space.

Algorithm 1: Approximate Inference Control with Regularized Update Steps

Input: initial state $\boldsymbol{q}_0$, parameter $\alpha^{[0]}$, threshold θ
Output: feedback control law $\boldsymbol{o}_{0:T-1}$ and $O_{0:T-1}$
initialize $\boldsymbol{q}_{1:T}^{[0]} = \boldsymbol{q}_0$, $S_0 = 1e10 \cdot \mathbf{I}$, $\boldsymbol{s}_0 = S_0 \boldsymbol{q}_0$, $n = 1$
while *not converged* **do**
  $\boldsymbol{q}_{0:T}^{[n-1]} = \boldsymbol{q}_{0:T}^{[n]}$
  for $t \leftarrow 1$ **to** T **do**
    linearize model: $A_t, \boldsymbol{a}_t, B_t$
    compute: $H_t, \boldsymbol{h}_t, R_t, \boldsymbol{r}_t$
    update: $\boldsymbol{s}_t$, S_t, $\boldsymbol{\nu}_t$, V_t, $\boldsymbol{b}_t$, and B_t
    if $\|\boldsymbol{b}_t - \boldsymbol{q}_t^{[n]}\| > \theta$ **then**
      repeat this time step
      $t \leftarrow t - 1$
    $\boldsymbol{q}_t^{[n]} = B_t^{-1} \boldsymbol{b}_t$
  for $t \leftarrow T - 1$ **to** 0 **do**
    ..same updates as above...
  for $t \leftarrow 0$ **to** $T - 1$ **do**
    compute feedback controller: $\boldsymbol{o}_t, O_t$
    $\boldsymbol{u}_t^{[n]} = \boldsymbol{o}_t + O_t \boldsymbol{q}_t$
    $\boldsymbol{q}_{t+1}^{[n]} = A_t \boldsymbol{q}_t^{[n]} + \boldsymbol{a}_t + B_t \boldsymbol{u}_t^{[n]}$
  $n = n + 1$
  $\alpha^{[n]} = \alpha^{[n-1]} \gamma$
return $\boldsymbol{o}_{0:T-1}$ and $O_{0:T-1}$

3.2.1 Evaluation of the convergence properties of AICO

To investigate the convergence properties of AICO, we use a simulated light-weight robot arm [28] with five joints. The robot has to reach a desired end effector position in Cartesian space and subsequently has to return to its initial pose. To increase the complexity, we define a second task, where the robot should additionally keep the roll angle of the end effector constant. For this task, we used the cost function

$$C_t(\boldsymbol{q}_t, \boldsymbol{u}_t) = \begin{cases} 10^4 (\boldsymbol{x}^i - \boldsymbol{x}_t)^T (\boldsymbol{x}^i - \boldsymbol{x}_t) & \text{if } t = T^i \\ 10^{-2} \boldsymbol{u}^T \boldsymbol{u} & \text{else} \end{cases}, \tag{3.8}$$

where $\boldsymbol{x}^i$ denotes the desired robot postures in task space at times $T^1 = 500$ and $T^2 = 10^3$ (the planning horizon is 2 seconds with a time step of 2ms) with $\boldsymbol{x}^1 = [1, -0.4, 0, 0, \pi/2, 0]^T$ and $\boldsymbol{x}^2 = [1, 0, 0, 0, \pi/2, 0]^T$. Note that we do not assume any initial solution to initialize the planner, solely the initial posture of the robot in configuration space is used as initial 'trajectory'. An example movement is shown in Figure 2.1.

Using the *pseudo-dynamics* approximation of the system dynamics, the convergence rate of the costs per iteration of both tasks are shown in Figure 3.1A,B. For the simple task in Figure 3.1A the inferred cost values converge fast for all algorithms, with the standard AICO algorithm showing the best performance. However, the fast convergence also comes with the costs of a reduced robustness of the policy update as can be seen from the results in the second scenario illustrated in Figure 3.1B, where AICO is unstable and cannot infer solutions with low costs. When we used the analytic forward dynamic model (where the linearizations are computed through finite differences) instead of the pseudo dynamics model, computing the messages in AICO became numerically unstable and no solutions could be inferred. Therefore, the panels in Figure 3.1C,D do not include results of AICO. We also evaluated the *iLQG* method [11] that implements an adaptive regularization schedule and line search to prevent divergence [12]. While the *iLQG* algorithm performed well for the pseudo dynamics model, the learning rate was automatically decreased to almost zero for the analytical dynamics model. Our regularization method for AICO, that we will present in the next section, considerably outperformed both competing methods.

3.3 Regulating the policy updates in AICO

To regularize the policy update steps in (3.1), we add an additional cost term to the task-likelihood function, i.e.,

$$p(r_t = 1|\boldsymbol{q}_t^{[n]}, \boldsymbol{u}_t^{[n]}) \propto \exp\{-C_t(\boldsymbol{q}_t^{[n]}, \boldsymbol{u}_t^{[n]}) - \alpha^{[n]}(\boldsymbol{q}_t^{[n]} - \boldsymbol{q}_t^{[n-1]})^T(\boldsymbol{q}_t^{[n]} - \boldsymbol{q}_t^{[n-1]})\} \ , \tag{3.9}$$

which punishes the distance of the state trajectories of two successive iterations of the algorithm (n denotes the iteration). The parameter α controls the size of the update step. For large α, the trajectory update will be conservative as the algorithm will stay close to the previous trajectory that has been used for linearization. For small α values, the new trajectory will directly jump to the LQG solution given the linearized dynamics and the approximated costs. Hence, α is inverse proportional to the step size. The value of α is updated after each iteration according to $\alpha^{[n]} = \alpha^{[n-1]}\gamma$. For $\alpha^{[0]} \geq 1$ and $\gamma > 1$, convergence is guaranteed as the regularization term will dominate with an increasing number of iterations.

The algorithms is listed in Algorithm 1. An interesting feature of this algorithm is that no learning rate is needed as α is used directly to implement a step size. In the original formulation of AICO the learning rate is either applied to the state update (in Line 13 in Algorithm 1) [13] or to the feedback controller (in Line 18 in Algorithm 1) [14]. However, neither implementation can guarantee convergence in nonlinear systems or in tasks with costs inducing a nonlinear mapping from $\mathbb{Q}$ to $\mathbb{X}$.

4 Experiments

We evaluate the resulting algorithm on the same robot arm reaching tasks. For both tasks, the Cartesian planning task in Figure 3.1A,C and the extension with the additional roll angle objective in Figure 3.1B,D, we evaluated AICO with the regularization parameter $\alpha \in \{1, 10\}$ (we did not increase α and $\gamma = 1$). For both models of the system dynamics, the *pseudo-dynamics* approximation (shown in Figure 3.1A,B) and the analytic model (illustrated in Figure 3.1C,D), AICO benefits from the regularization term and the costs decay exponentially fast. Interestingly, without "good" initial solutions, the differential dynamic programming method *iLQG* [12] that implements a sophisticated regularization scheme cannot generate movement policies with low costs when using the analytic model. This is shown in Figure 3.1C,D.

4.1 Simulation Results

We evaluated the proposed planning method in simulation with the humanoid robot Nao. The Nao robot has 25 degrees-of-freedom. In first experiments, we investigated the performance of the planner with a *pseudo-dynamics* model of the robot.
The humanoid had to reach for an end effector target using the right arm while maintaining balance. In a second experiment, Nao had to shift the x-coordinate of the center of gravity while maintaining balance.

4.1.1 Arm reaching with a humanoid robot

The humanoid has to reach for the end effector target $\boldsymbol{x}^* = [0, 0.2, 0.06]^T$, where only the y- and the z- Cartesian coordinates are relevant. Additionally, the robot has to maintain balance, which is implemented as deviation of the center of gravity vectors from its initial values $\boldsymbol{x}_{\text{CoG}}(t = 0)$, i.e., we specify the desired center of gravity as $\boldsymbol{x}^*_{\text{CoG}} = \boldsymbol{x}_{\text{CoG}}(t = 0)$. The same cost function as in the experiments for the light weight robot arm in (3.8) is used. For this task, however, only a single via-point is defined that is used for the desired end effector target and the center of gravity, i.e., $\boldsymbol{x}^1 = [\boldsymbol{x}^{*T}, \boldsymbol{x}^{*T}_{\text{CoG}}]^T$. Only by specifying two scalars in $\boldsymbol{x}^*$ (the scaling parameters in (3.8) are constants that take the values 10^4 or 10^{-2}), the planning algorithm infers 50-dimensional state trajectories (the state $\boldsymbol{q}_t$ at time t encodes the joint angles and the joint velocities, ignoring the base frame). This is shown in Figure 4.1A for the proposed planning algorithm with the regularization parameter $\alpha = 1$. As in the robot arm experiments, the Approximate Inference Control (AICO) algorithm

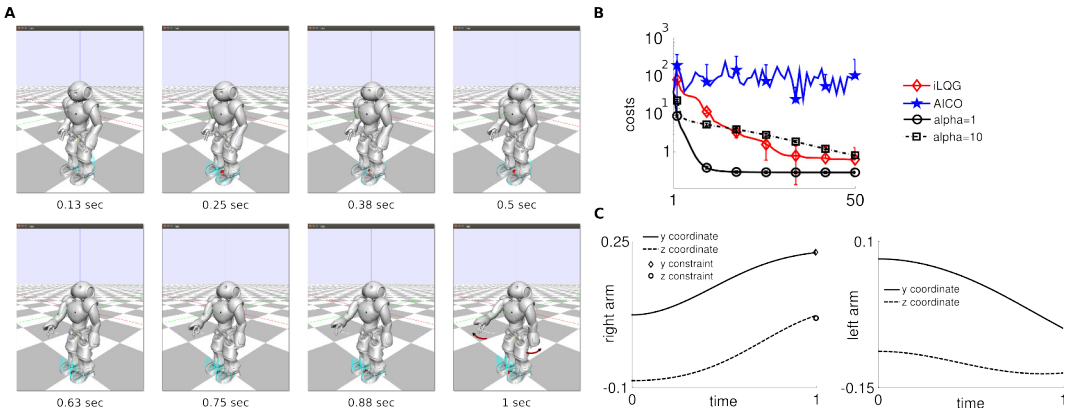


Figure 4.1: Reaching task with the humanoid robot *Nao*. The robot has to reach a desired end effector position with the right arm while maintaining balance. Eight snapshots of the inferred movement are shown in (A). In (B), the convergence of the costs of the optimization procedure is shown, where we compare *iLQG*, the standard implementation of AICO and the regularized variant. The mean and the standard deviations for 10 initial states '$\boldsymbol{q}_0$ are sampled from a Gaussian with zero mean and a standard deviation of 0.05. The movement objectives for the right arm are shown in the left panel in (C). To counter balance, the robot moves its left hand and the head.

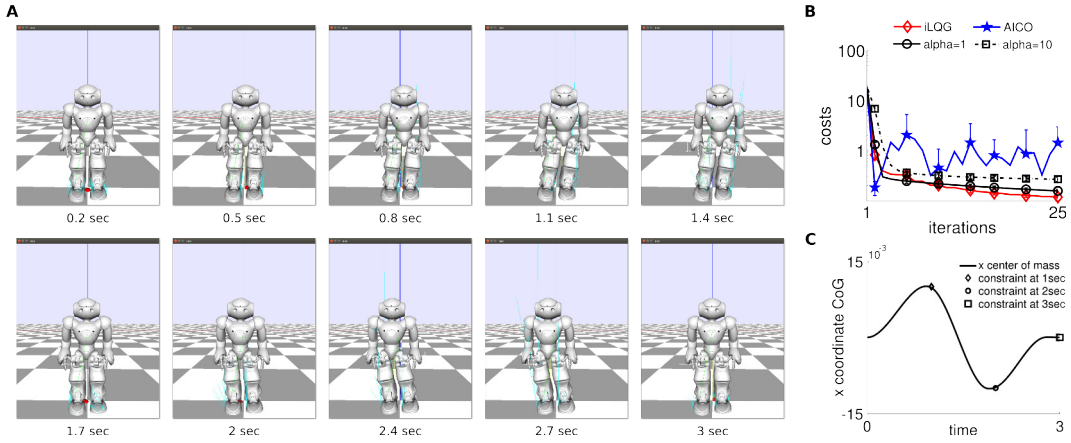


Figure 4.2: Balancing task in the humanoid robot *Nao*. The robot should *swing* its hips, which is encoded by adding an offset scalar to the x-coordinate of the center of gravity vector. In (A) 10 snapshots of the resulting movement for an increasing planning horizon are shown for $\alpha = 1$. The convergence properties of *iLQG*, the standard AICO and its regularized variants are shown in (B). The mean and the standard deviations for 10 initial states '$\boldsymbol{q}_0$ are sampled from a Gaussian with zero mean and a standard deviation of 0.05. In (C) the x-coordinate of the center of gravity of the Nao is illustrated. The large dots denote the objectives.

benefits from the regularization. As can be seen in Figure 4.1B, AICO cannot infer movement solutions with low costs without regularization.
Interestingly, to maintain balance, the humanoid utilizes its head and its left arm for which no objectives were explicitly specified. This effect is a feature of model-based planning methods that consider the coupled dynamics and is best illustrated in Figure 4.1C, where the end effector trajectories of both arms and the desired target values are shown.

4.1.2 Balancing with a humanoid

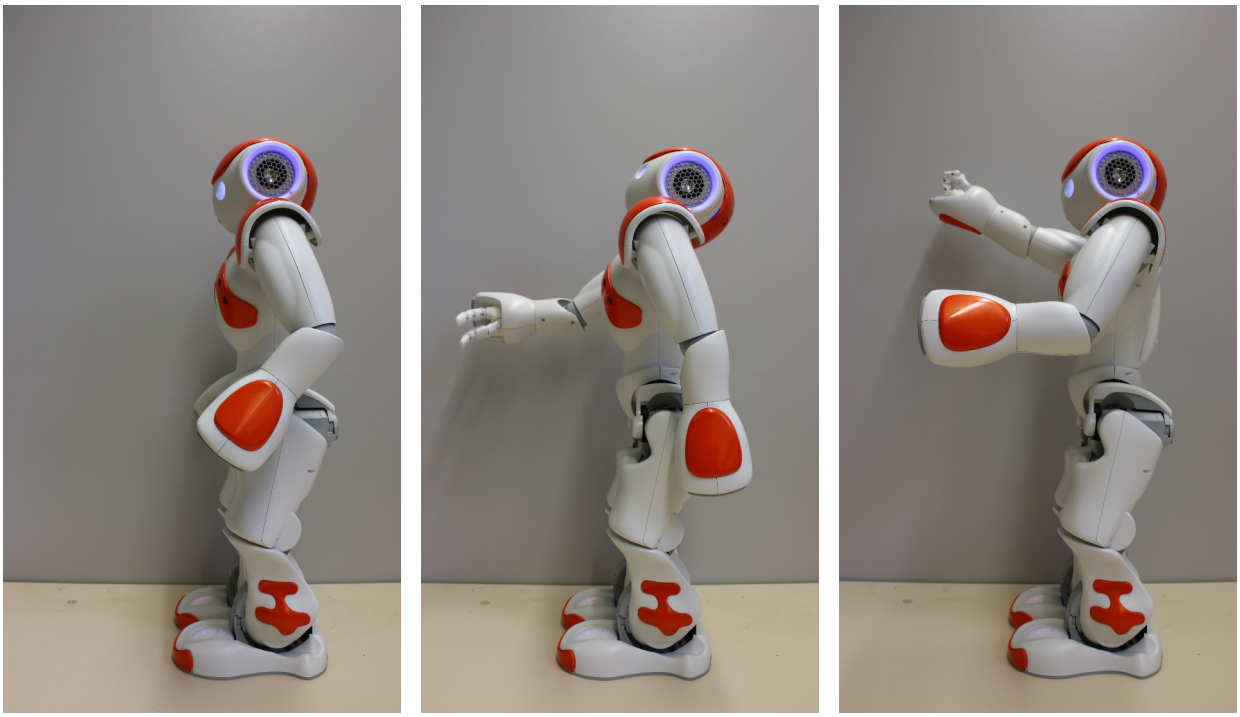
In this task the humanoid has to balance on one foot by moving its center of gravity. In this experiment, we specify three desired via-points for the center of gravity, i.e., $\boldsymbol{x}^i = \boldsymbol{x}^i_{\text{CoG}}$ with $i = 1, ..., 3$. The last via-point is set to the initial center of gravity $\boldsymbol{x}_{\text{CoG}}(t = 0)$. The first via-point has an offset of 0.1m in the x-coordinate of $\boldsymbol{x}_{\text{CoG}}(t = 0)$ to force the robot to move its center of gravity to the right. The second via point has the same negative offset in the x-direction to exhibit a movement to the left. The planning horizon was three seconds ($T^1 = 100, T^2 = 200$ and $T^3 = 300$ with $\tau = 10$ms) and the distance matrix C in (3.8) was scaled with the importance weights $[10^6, 10, 10]^T$ for the x,y, and z coordinate of $\boldsymbol{x}^i_{\text{CoG}}$.
For $\alpha = 1$, the resulting movement is illustrated in Figure 4.2A. Illustrated are 10 snapshots. Nao first moves its hip to the right (with respect to the robot frame) and thereafter to the left. This movement is the result of an inference problem encoded in mainly two scalars, i.e., the offsets.
The standard implementation of AICO was not able to infer successful balancing solutions, which is illustrated in Figure 4.2B. In contrast, the regularized variant using $\alpha \in \{1, 10\}$ converged after 25 iterations of the trajectory optimization procedure. For $\alpha = 1$, the x-coordinate of the center of gravity and the implemented objectives are shown in 4.2C.

4.1.3 Computational time

The computational time of the proposed planning algorithm is the same as for the standard implementation of AICO. If the algorithm is implemented in C-code it achieves real time performance in humanoid planning problems [13]. However, for our experiments we used a Matlab implementation on a standard computer (2.4GHz, 8GB RAM), where, e.g., the computation of the balancing movements in Figure 4.2 took less then 50 seconds (which includes all 25 iterations of the optimization process). The movement duration of the executed trajectory was three seconds.

4.2 Real Robot Results

Most stochastic optimal control approaches are only evaluated in simulations, with the exception of e.g. [3] (7-DoF robotic arm) and [1, 4] (7-Dof robotic arm with an attached hand). Hence, our planning method was also evaluated



(a) Initial position

(b) Arm Reaching Task (Video: `https://goo.gl/S7FRz1`)

(c) Joint Space Task (Video: `https://goo.gl/C7pxWW`)

Figure 4.3: Resulting end-positions of the learned movements applied to the real robotic platform Nao. The upper body is not moving, except of the desired movements, as the comparison to the initial position shows.

with the real robotic platform Nao to show the results gained in simulation also work in a real environment. Therefor, the planner is first trained with the *pseudo-dynamics* model of the robot and then evaluated on the real robot. As the simulated robot, the real Nao has 25 degrees-of-freedom.
The first task is the same as in 4.1.1, where the robot has to reach an end effector target using his right arm. In an additional experiment the Nao has to reach a given position in joint space while keeping balance.

4.2.1 Arm reaching with a humanoid robot

As a first experiment the results from 4.1.1 are applied to the real robot Nao. Again the robot has to reach for the end effector target $\boldsymbol{x}^* = [0, 0.2, 0.06]^T$, while maintaining balance. In Figure **??** the resulting movement is shown in comparison to the simulated movement. Without additional information, like friction models, the movement, shown in simulation, can be replicated on the real platform using the same planner. A video of the movement can be watched under `https://goo.gl/9Ic0qS`.

4.2.2 Joint Space Task

In this task the robot has to reach a certain position in joints space without moving its center of gravity (CoG). The robot has to raise both of his arms while turning the head and fixing the CoG. As joint position targets the following joint angles were specified as targets: The right shoulder pitch (RShoulderPitch) has to turn 1.2 rad, the right shoulder roll by 0.2 rad, the right elbow yaw turns 0.5 rad, the right elbow roll by -0.95 and the right wrist yaw by 1 rad and the left shoulder pitch has to turn 0.8 rad. The CoG should stay at its zero position during the whole movement, the difference of the CoG is weighted with 10^2 in every dimension. The cost function is similar to Equation 3.8

$$C_t(\boldsymbol{q}_t, \boldsymbol{u}_t) = \begin{cases} 10^4(\boldsymbol{q}^i - \boldsymbol{q}_t)^T(\boldsymbol{q}^i - \boldsymbol{q}_t) & \text{if } t = T^i \\ 10^{-2}\boldsymbol{u}^T\boldsymbol{u} & \text{else} \end{cases}, \tag{4.1}$$

where $\boldsymbol{q}^i$ denotes the desired robot postures in joint space at times $T^1 = 100$ (the planning horizon is 1 second with a time step of 1ms), whereas only the joint angles are considered in the cost function for which target positions are defined.

For the joints which have no defined target the current position is not considered in the cost function. The regularization parameter α is set to 1.
Figure 4.3c shows the resulting end position of the trajectory. A video of the movement can be watched in `https://goo.gl/C7pxWW`, where the simulation and the real robot is shown in parallel. For comparison here `https://goo.gl/xpmHE1` is the same target state of the robot but applied buy the SL goto funtion, without any constraints of the CoG. It can be seen that the robot is leaning forward, since the weight of the arms is pulling it. The optimization algorithm is preventing this behavior.

4.2.3 Computational time

For learning the showed trajectory with a duration of one second the Matlab implementation of the algorithm on a 2.6GHz Computer with 16GB RAM took about 3 minutes for 25 iterations.

5 Conclusion

Stochastic Optimal Control (SOC) methods are powerful planning methods to infer high-dimensional state and control sequences [11, 12, 13, 26]. For real time applications in humanoids, efficient model predictive control variants have been proposed [12]. However, the quality of the generated solutions heavily depends on the initial movement policy and on the accuracy of the approximations of the system dynamics. Most methods use regularization to prevent numerical instabilities, but typically greedily exploit the approximated system dynamics model. The resulting trajectory update might be far from the previous trajectory used for linearization.
As the linearizations are only locally valid, we explicitly avoid large jumps in the trajectories by punishing large deviations from the previous trajectory. We demonstrated in this paper that SOC methods can greatly benefit from such a regularization term. We used such regularization term for the Approximate Inference Control (AICO) algorithm [13]. Due to the regularization term, which implicitly specifies the step size of the trajectory update, no learning rate as in the standard formulation of AICO is needed. Our experiment shows that the used regularization term considerably outperforms existing SOC methods that are based on linearization, in particular if highly non-linear system dynamics are used.
An interesting open question is if the proposed regularization parameter facilitates a combination of SOC and model learning approaches. Typically, inaccurate model predictions have catastrophic effects on the numerical stability of SOC methods. In particular, if the model predictions are poor, the SOC method should not further explore but collect more data around the current trajectory. Such idea could be implemented by modeling the regularization parameter as a function of the model uncertainty.

5.1 Future Work on Nao Experiments

As future work it should be tried again to implement the previously failed approach mentioned in Section 1.1.4, i.e. appending the position and orientation of the base coordinate system to the state vector inside the optimization loop. As shown, using the SL simulation for computing the costs is not applicable. Providing the dynamics model with the base state vector seems to be the most promising approach in combination with SL. Because of the many other issues arisen during the project there was no time to implement and test this approach again.
As described in Section 1.1 there have been many problems posed by using SL. Consideration should be given using an other simulation software, with an active development of the community or some company and with a direct connection to Matlab, as for example V-Rep[1] or Webots[2], which both already have an integrated Nao model.

[1] http://www.coppeliarobotics.com/index.html
[2] https://www.cyberbotics.com/

Bibliography

[1] M. Toussaint, N. Plath, T. Lang, and N. Jetchev, "Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference," in *ICRA*, pp. 385–391, 2010.

[2] E. Rueckert, M. Mindt, J. Peters, and G. Neumann, "Robust policy updates for stochastic optimal control," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 388–393, Nov 2014.

[3] D. Mitrovic, S. Nagashima, S. Klanke, T. Matsubara, and S. Vijayakumar, "Optimal Feedback Control for anthropomorphic manipulators," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4143–4150, May 2010.

[4] F. Stulp, E. Theodorou, and S. Schaal, "Reinforcement Learning With Sequences of Motion Primitives for Robust Manipulation," *IEEE Transactions on Robotics*, vol. 28, pp. 1360–1370, Dec. 2012.

[5] P. Baerlocher and R. Boulic, "Task-priority formulations for the kinematic control of highly redundant articulated structures.," in *IROS*, pp. 13–17, 1998.

[6] S. I. Choi and B. K. Kim, "Obstacle avoidance control for redundant manipulators using collidability measure," *Robotica*, pp. 143–151, 2000.

[7] T. Sugihara and Y. Nakamura, "Whole-body cooperative balancing of humanoid robot using cog jacobian," in *IROS*, pp. 2575–2580, 2002.

[8] M. Gienger, H. Janssen, and C. Goerick, "Task-oriented whole body motion for humanoid robots," in *IEEE-RAS*, pp. 238–244, 2005.

[9] K. Nishiwaki, M. Kuga, S. Kagami, M. Inaba, and H. Inoue, "Whole-body cooperative balanced motion generation for reaching," *IJHR*, pp. 437–457, 2005.

[10] R. F. Stengel, *Stochastic optimal control: theory and application*. John Wiley & Sons, Inc., 1986.

[11] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *ACC*, pp. 300–306, 2005.

[12] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *IROS*, pp. 4906–4913, 2012.

[13] M. Toussaint, "Robot trajectory optimization using approximate inference," in *ICML*, pp. 1049–1056, 2009.

[14] E. Rueckert and G. Neumann, "Stochastic optimal control methods for investigating the power of morphological computation," *Artificial Life*, pp. 115–131, 2013.

[15] J. van den Berg, "Extended LQR: Locally-optimal feedback control for systems with non-linear dynamics and non-quadratic cost," in *Proc. Int. Symp. on Robotics Research*, 2013.

[16] W. Sun, J. v. d. Berg, and R. Alterovitz, "Stochastic Extended LQR: Optimization-Based Motion Planning Under Uncertainty," in *Algorithmic Foundations of Robotics XI* (H. L. Akin, N. M. Amato, V. Isler, and A. F. v. d. Stappen, eds.), no. 107 in Springer Tracts in Advanced Robotics, pp. 609–626, Springer International Publishing, 2015.

[17] F. Stulp, J. Buchli, E. Theodorou, and S. Schaal, "Reinforcement learning of full-body humanoid motor skills," in *2010 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 405–410, Dec. 2010.

[18] M. Toussaint and A. Storkey, "Probabilistic inference for solving discrete and continuous state markov decision processes," in *ICML*, pp. 945–952, 2006.

[19] M. D. Hoffman, N. D. Freitas, A. Doucet, and J. R. Peters, "An expectation maximization algorithm for continuous markov decision processes with arbitrary reward," in *AISTATS*, pp. 232–239, 2009.

[20] M. Toussaint and C. Goerick, "Probabilistic inference for structured planning in robotics," in *Int. Conf. on Intelligent Robots and Systems (IROS 2007)*, pp. 3068–3073, 2007.

[21] T. Furmston and D. Barber, "Variational methods for reinforcement learning," in *AISTATS*, pp. 241–248, 2010.

[22] M. Toussaint and C. Goerick, "A bayesian view on motor control and planning," in *From Motor Learning to Interaction Learning in Robots*, pp. 227–252, Springer, 2010.

[23] H. J. Kappen, V. Gómez, and M. Opper, "Optimal control as a graphical model inference problem," *JMLR*, pp. 159–182, 2012.

[24] J. Peters and S. Schaal, "Learning operational space control," in *Proceedings of Robotics: Science and Systems*, (Philadelphia, USA), August 2006.

[25] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *ICML*, pp. 745–750, 2007.

[26] K. Rawlik, M. Toussaint, and S. Vijayakumar, "An approximate inference approach to temporal optimization in optimal control," in *NIPS*, pp. 2011–2019, 2010.

[27] J. Van Den Berg, S. Patil, and R. Alterovitz, "Motion planning under uncertainty using iterative local optimization in belief space," *IJRR*, pp. 1263–1278, 2012.

[28] T. Lens, J. Kunz, O. v. Stryk, C. Trommer, and A. Karguth, "Biorob-arm: A quickly deployable and intrinsically safe, light-weight robot arm for service robotics applications," in *ISR/ROBOTIK*, pp. 1–6, 2010.