Sampling-Based Model Predictive Control with Neural Network Dynamics

Micah Reich
School of Computer Science
Carnegie Mellon University
mreich@cmu.edu

Jinhyung Park
Robotics Institute
Carnegie Mellon University
jinhyun1@andrew.cmu.edu

Jennifer Yang
Carnegie Institute of Technology
Carnegie Mellon University
jennifey@andrew.cmu.edu

Abstract—**Optimal control of robotic systems with non-linear dynamics and complex objective functions remains a challenging yet fundamental area of robotics research. We present a nonlinear Model Predictive Control (MPC) scheme for systems with unknown or complex dynamics by way of a neural dynamics model. Through the use of a Model Predictive Path Integral (MPPI) controller and sampling-based action sequence optimization, we demonstrate that neural network dynamics models can be substituted into traditional MPC schemes and achieve high performance in cost minimization. In addition, we show that neural networks can be used both within controllers as forward dynamics models as well as exterior to controllers as inverse dynamics models. Our code is available on GitHub.**

1. Introduction

Intelligent robotic machines repeatedly solve the sense, plan, act problem to achieve an autonomy goal. Within the "act" stage falls the study of robot control, or how to manipulate controlled variables within a robotic system to achieve a desired output.

1.1. Robot Control Algorithms

There exist many paradigms of robot control, where schemes often depend on the linearity, stability, or time-invariant nature of a robotic system.

Feedforward control schemes use knowledge of plant dynamics to calculate control inputs which result in a desired system trajectory. Feedback control schemes use feedback, or an observation of the system state, to drive the system along a reference trajectory. Feedback-feedforward control schemes combine both methods, using a model of the system to derive an initial control, using state observations to ensure the system steady-state converges to the reference trajectory.

Feedback-feedforward control schemes vary in complexity, ranging from a simple LTI plant model coupled with a PID controller, to schemes like non-linear Model Predictive Control (MPC). MPC is a highly versatile control algorithm which, unlike other cost-minimizing control strategies like LQR, can operate agnostic to both the plant linearity and the form of cost functions [3], [4].

1.2. System Modelling

1.2.1. Dynamics Models. For feedforward control schemes which require a plant model, it is common to use a kinematic or dynamic model of the system. Using Lagrange's Method or the Newton-Euler formulation, the equations of motion (EOMs) of tractable systems may be derived. The forward dynamics yield system accelerations as a result of input wrenches, and inverse dynamics determine requisite wrenches needed to achieve desired system accelerations.

1.2.2. Neural Networks. In recent years, the acceleration of GPU computing and deep learning research has rapidly advanced the state and performance of neural networks. As universal function approximators, neural networks have widely been used for language, vision, and robotics tasks. In this work, we leverage a neural network to model dynamics forward transitions, which allows us the flexibility to control systems in the wild for which it may be difficult to derive a precise dynamics model.

2. Methods

2.1. Model Predictive Control

MPC [3] is a receding-horizon optimal control scheme for systems with linear or nonlinear dynamics. Using a model of the system, MPC repeatedly solves a constrained optimization problem of the form:

$$\min_{\mathbf{u}} \quad \phi(\mathbf{x}(T), \mathbf{u}(T)) + \sum_{k=t}^{t+T} J(\mathbf{x}(k), \mathbf{u}(k))$$

such that

$$\begin{aligned} \mathbf{x}(t+1) &= f(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{u}_{lb} \leq \mathbf{u}&(\cdot) \leq \mathbf{u}_{ub} \\ \mathbf{x}_{lb} \leq \mathbf{x}&(\cdot) \leq \mathbf{x}_{ub} \end{aligned}$$

where J is the stage cost function over the states and control inputs, and ϕ is a terminal cost over the final state and control input in the horizon.

Traditional MPC discretizes the action space for a fixed number of simulated time steps into the future, known as the horizon length, and performs a tree search along the temporal discretized action space to determine the optimal sequence of actions to execute. The first element of this action sequence, $\mathbf{u_0}$, is executed on the real (or simulated) system, then the OCP is solved again at the next time step.

Due to the exponentially-growing nature of the size of a traditional MPC rollout, horizon lengths are generally kept small. In addition, depending on the complexity of the model used, MPC schemes are typically computationally expensive. Luckily, modern compute hardware like GPUs allow for fast, online, parallelized computation.

2.2. Model Predictive Path Integral Control

MPPI [7] is an MPC scheme that uses Monte-Carlo sampling of controls $\hat{\mathbf{u}}$ to approximate a control sample $\mathbf{u} \sim Q$ from the time-horizon optimal control distribution Q. MPPI uses K independent rollouts and a time horizon, T to explore the state space with sampled control sequences, choosing a sequence (or weighted average of sequences) which yield low cost trajectories. This results in a versatile scheme which does not impose particular requirements (i.e. convex costs, linear dynamics) on the optimization problem.

This paper's MPPI implementation assumes an optimization problem of the form:

$$\min_{\mathbf{u}} \quad \phi(\mathbf{x}_T) + \sum_{k=t}^{t+T} q(\mathbf{x}_k) + r(\hat{\mathbf{u}}_k)$$

with discrete-time dynamics, where q and r are state and control stage costs, respectively.

For K and a horizon T, at each time step, MPPI first generates random control perturbations

$$\delta^i = (\delta^i_1, \delta^i_2, \cdots, \delta^i_T) \quad \delta^i_t \sim \mathcal{N}(0, \Sigma_t)$$

for each rollout $1 \leq i \leq K$. Given a previously-computed nominal control sequence $\mathbf{u} = (u_1, u_2, \cdots, u_T)$ and covariance sequence $\mathbf{\Sigma} = (\Sigma_1, \Sigma_2, \cdots, \Sigma_T)$, the sampled controls for rollout i are given as:

$$\hat{\mathbf{u}}^i = (u_1 + \delta^i_1, u_2 + \delta^i_2, \cdots, u_T + \delta^i_T)$$

MPPI then propagates each rollout state forward, using

$$\mathbf{x}^i_{t+1} = f(\mathbf{x}^i_t, \hat{\mathbf{u}}^i_t)$$

for $t = t_0, t_0 + 1, \cdots, t_0 + T$. The cost of rollout i, C_i, is computed as:

$$C_i = \phi(\mathbf{x}^i_T) + \sum_{k=t}^{t+T} \mathbf{q}(\mathbf{x}^i_k) + \mathbf{r}(\hat{\mathbf{u}}^i_k)$$

and the score of rollout i, S_i, is computed as:

$$S_i = \exp\left(-\frac{1}{\lambda}(C_i - \beta)\right) \text{ where } \beta = \min_i \ C_i, \quad w_i = \frac{S_i}{\eta}$$

for a constant scalar λ, a temperature parameter similar to that of a softmax function, where $\eta = \sum_{i=1}^{K} S_i$ is a normalization constant.

A new nominal control and covariance sequence, $\mathbf{u}'$ and $\mathbf{\Sigma}'$, must now be computed using the costs associated with sampled control sequences $\mathbf{u}_i$. This is achieved by computing these new sequences as [2]:

$$\mathbf{u}'_j = \alpha_\mu \mathbf{u}_j + (1 - \alpha_\mu)\left(\sum_{i=1}^{K} w_i \hat{\mathbf{u}}^i_j\right)$$
$$\mathbf{\Sigma}'_j = \alpha_\sigma \mathbf{\Sigma}_j + (1 - \alpha_\sigma)\left(\sum_{i=1}^{K} w_i (\hat{\mathbf{u}}^i_j - \mathbf{u}^i_j)(\hat{\mathbf{u}}^i_j - \mathbf{u}^i_j)^\top\right)$$

where α_μ and α_σ are parameters which vary the similarity of new and previously computed controls. After the new nominal control sequence $\mathbf{u}'$ is computed, $\mathbf{u}'_1$ is sent to the real or simulated system actuators. Finally, $\mathbf{u}'$ is shifted forward in time by 1 time step, and the last element, $\mathbf{u}'_T$, is initialized to a fixed or random value.

2.3. Neural Network Dynamics Models

Traditionally, MPC uses a derived dynamics model for the state transition, $\mathbf{f}(\cdot)$, or for the inverse dynamics, $\mathbf{f}^{-1}(\cdot)$. We demonstrate the ability to instead model $\mathbf{f}(\cdot)$ and $\mathbf{f}^{-1}(\cdot)$ as neural networks based on Williams et al. [7], [8], [9].

To train forward dynamics models, we collect sequences of $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$ triplets as training data [7], [8], [9]. The model takes as input $[\mathbf{x}_t, \mathbf{u}_t]$ and outputs $\mathbf{f}(\cdot) = \ddot{\mathbf{x}}_t$, thus:

$$\mathbf{x}_{t+1} = \begin{bmatrix} \mathbf{x}_t + \dot{\mathbf{x}}_t \Delta t \\ \dot{\mathbf{x}}_t + \mathbf{f}(\cdot)\Delta t \end{bmatrix}$$

Algorithm 1 Data generation for forward dynamics models

```
Require: $N > 0, T > 0$
  $K \leftarrow N/T$
  $\mathbf{x}^k(0) \leftarrow$ RANDOM STATE for $(k \leq K)$
  for $t \leftarrow 1, T$ do
      for $k \leftarrow 1, K$ do
          $\mathbf{u} \leftarrow$ RANDOM ACTION
          $\mathbf{x}^k_{t+1} \leftarrow \mathbf{f}(\mathbf{x}^k_t, \mathbf{u})$
      end for
  end for
```

for a small time step Δt.

In our experiments, we train small neural networks with 2 fully connected layers, each ranging from 16 to 128 nodes. The network sizes are kept small to minimize inferencing times, as the models are run inside the controllers online at each time step. Our technique for data generation is shown in Algorithm 1, where T is a horizon length and N is the number of triplets. This strategy is employed to reduce the multi-step prediction error from [8], [9].

To train inverse dynamics models, we collect sequences of $(\mathcal{F}_t, \ddot{\mathbf{x}}_t)$ pairs as training data. The model takes as input $[\ddot{\mathbf{x}}]$ and outputs $\mathbf{f}^{-1}(\cdot) = \mathcal{F}$, where $\mathcal{F}$ is a torque or force vector.

3. Results

We examine the performance of MPPI with ground-truth dynamics and neural dynamics on three dynamic systems: a damped pendulum, cart-pole, and planar two-revolute joint (RR) arm. For each system, we derive the EOMs and run identically-parameterized controllers, where one uses the ground truth EOMs, denoted as C_G, and the other uses a neural network, denoted as C_N.

In all cost functions, $Q, R,$ and M refer to diagonal weighting matrices, and all systems use a quadratic control cost:

$$r(\mathbf{u}) = u^\top R u$$

3.1. Pendulum

The state-space representation of this system is $q = [\theta, \dot{\theta}]$ with the control input as torque τ. The EOMs are given by [5]:

$$\ddot{\theta} = \frac{\tau - mgl\sin(\theta) - b\dot{\theta}}{ml^2}$$

where b is a damping coefficient. The MPPI controllers directly compute the optimal torque needed to achieve the desired state, namely $\theta_d = \pi$, $\dot{\theta}_d = 0$. The cost function of the controller is of the form:

$$q(\mathbf{x}) = x_e^\top Q x_e, \quad x_e = [(\cos(\theta_d - \theta) - 1, \dot{\theta}_d - \dot{\theta}]$$
$$\phi(\mathbf{x}) = 0$$

The system is torque-limited to ± 2 Nm, requiring the controller to build momentum to reach the desired state. As seen in Figure 1, starting from rest, the controllers converge on different control solutions, both reaching the desired state by $10s$. C_N is able to reach the set point faster than C_G by around $1s$. In our experiments, we find that C_G is usually as fast or faster than C_N, with C_N sometimes requiring an extra cycle of pendulum oscillations before reaching the set point at steady-state.

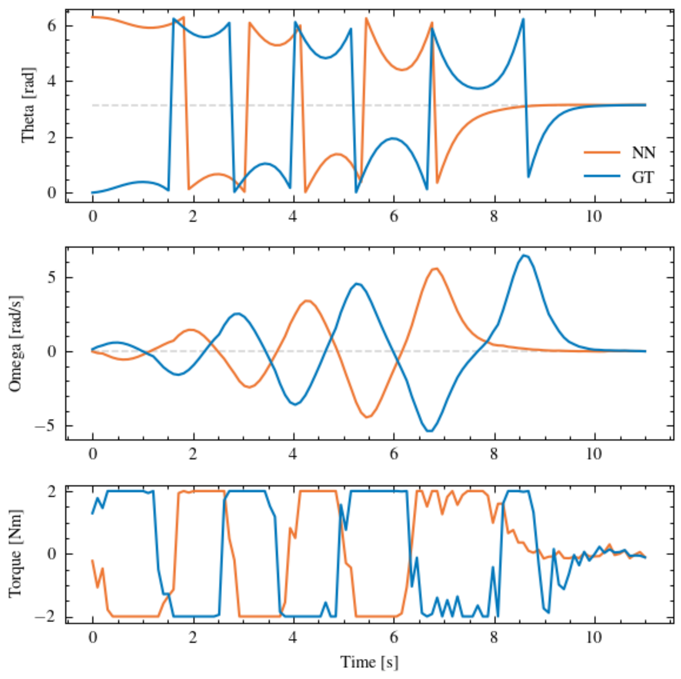

Figure 1. Pendulum swing up task with $m = 1$kg, $l = 1$m. Results shown for one controller using neural dynamics and the other using ground-truth pendulum EOMs (GT).

3.2. Cart-Pole

The state-space representation of this system is $q = [x, \dot{x}, \theta, \dot{\theta}]$ with the control input as force acting on the cart along the x-direction, f_x. The EOMs are given by [5]:

$$\ddot{x} = \frac{f_x + m_p \sin(\theta)(l\dot{\theta}^2 + \cos(\theta))}{m_c + m_p \sin^2(\theta)}$$

$$\ddot{\theta} = \frac{-f_x \cos\theta - m_p l\dot{\theta}^2 \cos(\theta)\sin(\theta) - (m_c + m_p)g\sin(\theta)}{l(m_c + m_p \sin^2(\theta))}$$

Similar to the pendulum's cost, the cart-pole controller cost function seeks to stabilize the system around $\theta_d = \pi$, $\dot{\theta}_d = 0$, and $x = 0$:

$$q(\mathbf{x}) = x_e^\top Q x_e, \quad x_e = [(\cos(\theta_d - \theta) - 1, x_d - x]$$
$$\phi(\mathbf{x}) = x_t^\top M x_t, \quad x_t = [(\cos(\theta_d - \theta) - 1, \dot{\theta}_d - \dot{\theta}, x_d - x]$$

The system is force limited to ± 10 Nm, demanding similar behavior out of the controller to that of the torque-limited pendulum. As seen in Figure 2, C_G reaches the setpoint around 0.4s faster than C_N, and the behavior of both controllers is similar. In our experiments, we find that the only significant difference between C_G and C_N is the ability to stabilize around $x = 0$. This discrepancy is likely due to an observed compounding position prediction error by the neural network.

3.3. Planar 2R Arm

The state-space representation of this system is $q = [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2]$ with the control input as $\tau = [\tau_1, \tau_2]$, torques sent to the joint actuators. The dynamics of this system follow from the manipulator EOMs as given by [5]:

$$\mathbf{M}(q)\ddot{q} + \mathbf{C}(q, \dot{q})\dot{q} = \tau_g(q) + \mathbf{B}\tau$$

The cost function for the controllers consists of a simple path deviation term using the arm's forward kinematics, FK:

$$q(\mathbf{x}) = x_e^\top Q x_e, \quad x_e = ||\rho_t - \text{FK}(\vec{\theta})||$$
$$\phi(\mathbf{x}) = \dot{\theta}^\top M \dot{\theta}$$

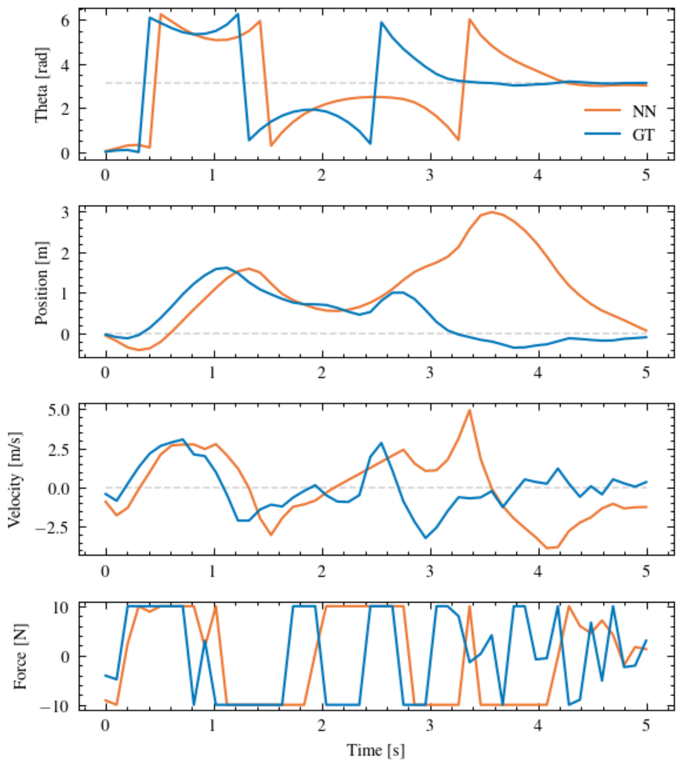

Figure 2. Cart-pole swing up task with $m_c = m_p = 1$kg, $l = 1$m. Results shown for two MPPI controllers, one using neural dynamics for rollouts (NN) and the other using the ground-truth cart-pole EOMs (GT).

where $\rho(\cdot)$ is a parametric curve.

Rather than end-to-end torque control by the MPPI controller, we employ feedback linearization of the form $u = [\ddot{\theta}_1, \ddot{\theta}_2]$. The controller computes joint accelerations and the manipulator inverse dynamics are used to compute the joint torques needed to achieve these commands.

The planar 2R arm was tested, with gravity acting in the $-y$ direction, on point-stabilization tasks as well as Cartesian path tracking. Figure 3 shows a parametric lemniscate path and the arm's tracking performance. For tracking tasks, the system's state is modified to include the current value of the path parameter. In our experiments, both controllers perform similarly, with an average deviation of 50-80mm from the desired path.

3.4. Mean Trajectory Costs

We summarize the relative performance of C_G versus C_N by examining the ratio of mean trajectory costs for the above tasks.

TABLE 1. RATIO OF MEAN TRAJECTORY COSTS

System	Task	Cost(C_N) / Cost(C_G)
Pendulum	Swing Up	0.993
Cart-Pole	Swing Up	1.026
2R Arm	Stabilization	1.037
	Path Tracking	0.998

Each task was run 20 times. For each trial, the state and control stage costs were computed on the current system state and executed control, and the terminal cost was computed once at the end of each trial.

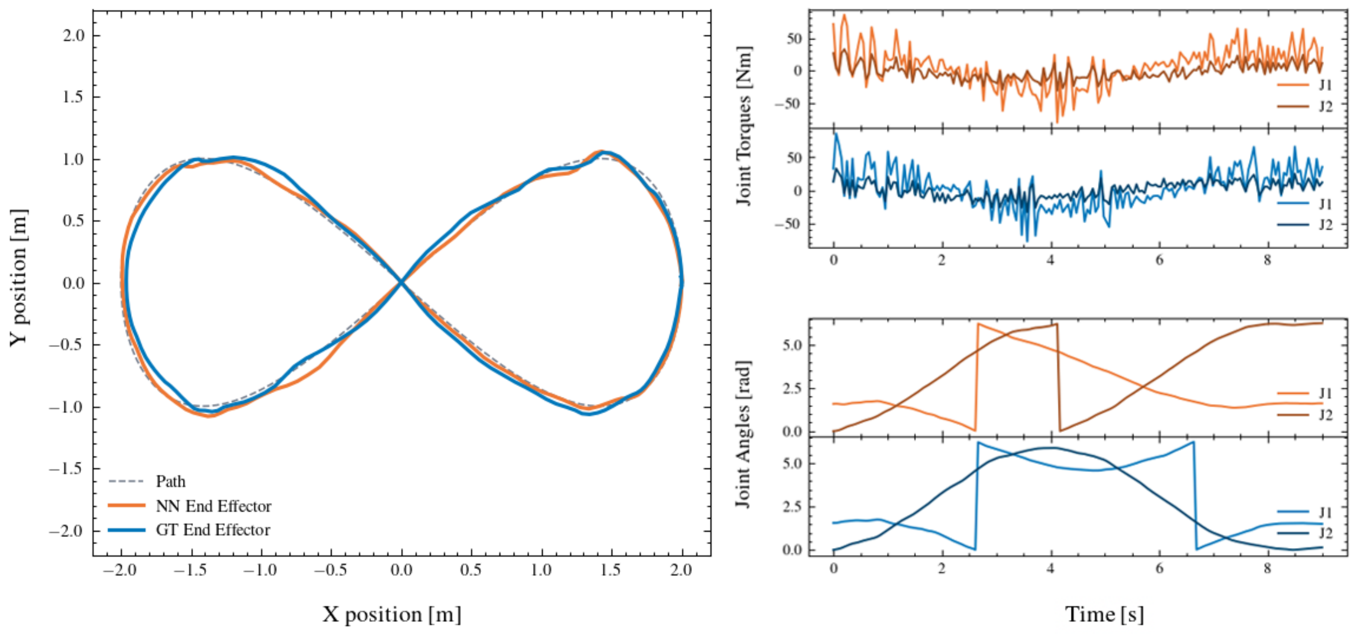

Figure 3. 2R arm path tracking task with $m_1 = m_2 = 1$kg, $l_1 = l_2 = 1$m. Results shown for two MPPI controllers tracking a lemniscate path, one using neural inverse dynamics (NN) and the other using the ground-truth 2R arm inverse dynamics (GT). The path is specified in Cartesian coordinates.

4. Discussion

4.1. Comparison to Prior Work

We train the neural network dynamics on a time horizon of one second. However, prediction errors compound over multiple steps. This is show in Figure 4, where the multi-step prediction error (RMSE) is shown for the linear position and angle of the cart-pole. For position, the error when trained on a 16 node model grows much faster than the error when trained on 32 or 64 nodes. For angle, we see similar growth in the error of the 16 node model with respect to the 32 and 64-node models. Compared to work done by Williams, the position error of our model is similar but slightly higher, though the angle error is far less than that of Williams' models [9]. The multi-step error increasing exponentially may be overcome by using alternative training schemes.

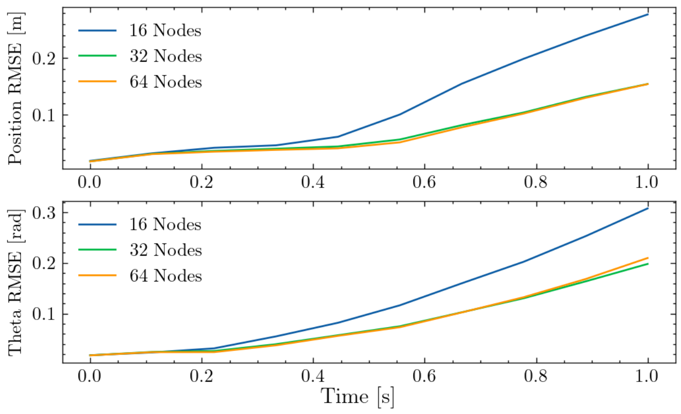

Figure 4. Multi-step prediction error (RMSE) on Cart-Pole position and pendulum angle for 3 neural network sizes. Each network is trained on $36,000$ samples, e.g. 60 mins of data at 10Hz.

4.2. Future Work

4.2.1. Disturbance Rejection. In regions of the state-space with low cost, the controller's covariance sequences will tend towards zero in the limit. This poses an issue for disturbance rejection, as perturbations of the system while in a steady-state will be difficult to overcome due to low sampling covariance. This may be remedied by injecting additional rollouts with high sampling covariance at every time step, giving the controller the freedom to choose distant samples if needed.

4.2.2. Online Learning. Our neural network dynamics model accurately captures the true dynamics to support MPPI rollouts. However, its performance under testing distributions with deviations from the training domain remains untested. In future work, we plan to incorporate elements of domain randomization [6] and online learning [1] to both allow our neural dynamics model to be more robust to environment changes and to enable our model to quickly adapt to new transitions and situations on the fly.

4.2.3. Extension to Real-Life Systems. Although we try to account for variations and disturbances, the neural dynamics models are currently not verified with real-life systems. In the future, we first intend to verify our work with simple models of the pendulum, cart-pole, and planar 2R arm. Then, we intend extend these models to more complex systems, such as arms with more degrees of freedom or submersible robots.

References

[1] Terry Anderson. *The theory and practice of online learning*. Athabasca University Press, 2008.

[2] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *Conference on Robot Learning*, pages 750–759. PMLR, 2022.

[3] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[4] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229. Citeseer, 2004.

[5] Russ Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for mit 6.832. *Working draft edition*, 3:4, 2009.

[6] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[7] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

[8] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018.

[9] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.