A Generalized Importance Sampling Method for Model Predictive Path Integral Control

Grady Williams, Andrew Aldrich and Evangelos A. Theodorou
Autonomous Control and Decisions Systems Laboratory
Institute for Robotics and Intelligent Machines
Georgia Institute of Technology, Atlanta, Georgia

Abstract

In this paper we develop a Model Predictive Path Integral (MPPI) control algorithm based on a generalized importance sampling scheme and perform parallel optimization via sampling using a Graphics Processing Unit (GPU). The proposed generalized importance sampling scheme allows for changes in the drift and diffusion terms of stochastic diffusion processes and plays a significant role in the performance of the algorithm. The resulting algorithm is able to efficiently guide teams of quadrotors through cluttered simulated environments.

1 Introduction

Path integral control [1, 2, 3, 4, 5] is a powerful method for stochastic trajectory optimization which enables the mathematically rigorous development of sampling based control algorithms. The key idea in the path integral control framework is that the value function is specified using the Feynman-Kac lemma [6, 7, 8]. The use of the Feynman-Kac lemma allows stochastic optimal control problems to be solved with forward sampling of stochastic diffusion processes and evaluations of expectations along these sampled trajectories. This approach is in stark contrast to classical optimal control approaches [9], [10] which use methods which are backwards in time. In the path integral control framework we consider stochastic optimal control problems of the form:

\[ V(x, t) = \min_u E \left[ \phi(x_T) + \int_{t_0}^T \left( q(x, t) + \frac{1}{2} u^T R u \right) dt \right] \] (1)

where \( V(x, t) \) is the value function, \( \phi(x_T) \) is the terminal cost, \( q(x, t) \) is state dependent running cost, \( R \) is a positive definite matrix and \( T \) is the finite time horizon. The cost function under minimization is subject to the dynamics:

\[ dx = f(x, t) dt + G(x, t) u(x, t) dt + B(x, t) dw \] (2)

where \( f(x, t) \) is the drift term, \( G(x, t) \) is the control transition matrix and \( B(x, t) \) is the diffusion matrix. In the following presentation we drop the functional dependence for notational convenience. The approach is to apply the exponential transformation of the value function \( V(x, t) = -\lambda \log(\Psi(x, t)) \) and assume that \( BB^T = \lambda GR^{-1} G^T \). In this case \( \Psi(x, t) \) satisfies the backward Chapman Kolomogorov PDE:

\[ \partial_t \Psi = \frac{\Psi}{\lambda} q - f^T \nabla_x \Psi - \frac{1}{2} \text{tr}(BB^T \nabla_{xx} \Psi) \] (3)

Which is a linear PDE in terms of \( \psi \). The assumption that \( BB^T = \lambda GR^{-1} G^T \) implies that the magnitude of the noise in a particular state is proportional to the direct control authority over that state. In other words, if a state suffers from a high amount of noise it must also be cheap to control. The Feynman-Kac theorem relates backwards PDE’s of this type to path integrals of the form:

\[ \Psi(x_{t_0}, t_0) = E_p \left[ \exp \left( -\frac{1}{\lambda} \int_{t_0}^T q(x, t) \ dt \right) \Psi(x_T, T) \right] \] (4)

1
where the term $\Psi(x_{k}, t_0)$ is the transformed terminal cost: $e^{-\frac{1}{2}\phi(x_T)}$, and by discretizing time we can re-write this expression as:

$$
\Psi(x_{k}, t_0) \approx \mathbb{E}_p \left[ \exp \left( -\frac{1}{\lambda} \left( \phi(x_T) + \sum_{t_0}^{N} q(x_t) \, dt \right) \right) \right] 
$$

where $N = \frac{T-t_0}{dt}$. Lastly we have to compute the gradient of $\Psi$ with respect to the initial state $x_0$ as well. Doing this yields:

$$
u^* = R^{-1}G^TH^{-1} \int_\tau^\infty \frac{\exp \left( -\frac{1}{\lambda} S(\tau) \right) p(\tau) z_t}{\int_\tau^\infty \exp \left( -\frac{1}{\lambda} S(\tau) \right) p(\tau) d\tau} d\tau 
$$

where $S(\tau) = \phi(x_T) + \sum_{t_0}^{N} q(x_t) \, dt$ is the state dependent portion of the cost-to-go, $x^c_t$ denotes the directly actuated component of the state at time $t$, the matrix $H = GR^{-1}G^T$, and $p(\tau)$ is the probability of a trajectory under the discrete time uncontrolled stochastic dynamics: $dx = f(x, t) dt + B dw$. We can compute this probability by conditioning and using the Markov property of the state space to get: $p(\tau) = p(x^c_t, \ldots, x^c_{t_0} | x_{t_0}) = \prod_{i=1}^{N} p(x^c_t | x_{t_{i-1}})$. Note that only the directly actuated states appear as random variables in this equation since (by our assumption that $BB^T = \lambda GR^{-1}G^T$) the non-directly actuated component of the state space is deterministic. The one-step probability $p(x^c_t | x_{t_0})$ is Gaussian with mean $f(x,t)$ and variance $\Sigma = B B^T$. Letting $\mu(x_t)$ denote the parts of $f$ and $B$ associated with $x_c$. By incorporating the transition probabilities the probability of the trajectory $\tau$ is expressed as follows:

$$p(\tau) = \exp \left( -\frac{1}{2} \sum_{i=1}^{N} \left( \mu(x_{t_i}) - f_c(x_{t_{i-1}}) \right)^T \Sigma^{-1} \left( \mu(x_{t_i}) - f_c(x_{t_{i-1}}) \right) \right) \prod_{i=1}^{N} (2\pi)^{n/2} \det(\Sigma)^{(1/2)} 
$$

where the term $\mu(x_{t_i})$ is defined as $\mu(x_{t_i}) = \frac{(x^c_{t_i} - x^c_{t_0})}{dt}$.

2 Generalized Importance Sampling.

The major issue with the expression for the optimal controls derived in the previous section is that sampling from the uncontrolled dynamics can be very inefficient. Iterative versions of PI control using importance sampling have been derived in [3, 4, 11]. However, these methods only account for shifting the mean of the distribution. In practice it is often necessary to change the variance of the distribution as well since in most control systems random variations in the natural system aren’t sufficiently large enough to produce interesting behavior. The method we derive here accounts for both the mean and variance. We begin our analysis by considering the PI form of the optimal controls developed in the previous section:

$$u_{t_i}^* = R^{-1}G_c(x_{t_i})^TH(x_{t_i})^{-1} \int_{\tau_{i}}^{\infty} \frac{\exp \left( -\frac{1}{\lambda} S(\tau_i) \right) p(\tau_i) \left( \frac{x_{t_{i+1}} - x_{t_i}}{dt} - f(x_{t_i}) \right)}{\int_{\tau_{i}}^{\infty} \exp \left( -\frac{1}{\lambda} S(\tau_i) \right) p(\tau_i) d\tau_i} d\tau_i 
$$

where $p(\tau_i)$ is the probability of a trajectory under the uncontrolled dynamics of the system and $S(\tau_i) = \phi(x_{t_i}) + \sum_{t_0}^{N} q(x_{t_i}) \, dt$ is the state dependent portion of the cost-to-go. Letting $z_{t_i} = \frac{x_{t_{i+1}} - x_{t_i}}{dt} - f(x_{t_i})$ and re-writing the integral as the ratio of two expectations yields:

$$u_{t_i}^* = R^{-1}G_c(x_{t_i})^TH(x_{t_i})^{-1} \left( \frac{\mathbb{E}_p \left[ \exp \left( -\frac{1}{\lambda} S(\tau_i) \right) z_{t_i} \right]}{\mathbb{E}_p \left[ \exp \left( -\frac{1}{\lambda} S(\tau_i) \right) \right]} \right) 
$$

This form suggests a simple method to approximate optimal controls. We know by the strong law of large numbers that as $K \to \infty$:

$$R^{-1}G_c(x_{t_i})^TH(x_{t_i})^{-1} \left( \frac{1}{K} \sum_{k=1}^{K} \exp \left( -\frac{1}{\lambda} S(\tau_i^k) \right) z_{t_i} \right) \to u^*_{t_i} 
$$
In order to get a good estimate of \( u_t^* \), it’s necessary to sample from a distribution which has a higher probability of generating useful trajectories than the uncontrolled dynamics. There is a well known trick in the Monte-Carlo literature to change a sampling distribution from a poorly conditioned one like \( p(\tau_t) \) to one which is more likely to produce valuable samples. Let \( q(\tau_t) \) be the probability of a trajectory under a new diffusion process which has a non-zero control input and a changed variance. We can then express the optimal controls as an expectation under \( q(\tau_t) \) instead of \( p(\tau_t) \):

\[
u_t^* = R^{-1} G_c(x_{t_i})^T H(x_{t_i})^{-1} \left( \frac{E_q \left[ \exp \left( -\frac{1}{\lambda} S(\tau_i) \right) \frac{p(\tau_t|x_{t_i})}{q(\tau_t)} z_{t_i} \right]}{E_q \left[ \exp \left( -\frac{1}{\lambda} S(\tau_i) \right) \frac{p(\tau_t)}{q(\tau_t)} \right]} \right)
\]

(11)

The advantage is that we can now engineer the mean and variance of \( q(\tau_t) \) to produce the best results possible. However, we also have an extra term to compute: \( \frac{p(\tau_t)}{q(\tau_t)} \) this is known as the Radon-Nikodym derivative (or the likelihood ratio) [7] between the two distributions \( p(\tau_t) \) and \( q(\tau_t) \). We now derive this term for path integral case. We consider the two diffusion processes defined by:

\[
dx = f_c(x_t)dt + B^c_t(x_t)dw \quad \text{and} \quad \dx = f_c(x_t)dt + G(x_t)u_t dt + B^e_t(x_t)dw
\]

(12)

where \( B^c_t(B^c_t)^T = \Sigma_1 \) and \( B^e_t(B^e_t)^T = \Sigma_2 \). Here \( \Sigma_1 \) is the variance of the natural process, and \( \Sigma_2 \) is the exploration variance which is a parameter of the algorithm. Note that in general \( \Sigma_1 \) and \( \Sigma_2 \) can be functions of both the state and time, however for the rest of our analysis we assume that \( G, B^c, B^e \) are state independent. In discrete time the probability of a trajectory \( p(\tau_i) = p(x_{t_i}, x_{t_{i+1}}, \ldots x_{t_N}) \) is formulated according to the previous section:

\[
p(\tau_i) = Z_i^{-1} \exp \left\{ \sum_{j=1}^{N} -\frac{1}{2} (z_{t_j}^T \Sigma_1^{-1} z_{t_j}) \right\}, \quad q(\tau_i) = Z_2^{-1} \exp \left\{ \sum_{j=1}^{N} -\frac{1}{2} ((z_{t_j} - \mu_{t_j})^T \Sigma_2^{-1} (z_{t_j} - \mu_{t_j})) \right\}
\]

(13)

where \( \mu_{t_j} = Gu_{t_j} \). Then \( \frac{p(\tau_i)}{q(\tau_i)} \) is equal to:

\[
\frac{p(\tau_i)}{q(\tau_i)} = Z_2 Z_i^{-1} \exp \left\{ -\frac{d\tau}{2} \sum_{j=1}^{N} \zeta_{t_j} \right\}, \quad \text{where} \quad \zeta_{t_j} = (z_{t_j}^T \Sigma_1^{-1} z_{t_j} - (z_{t_j} - \mu_{t_j})^T \Sigma_2^{-1} (z_{t_j} - \mu_{t_j}))
\]

(14)

Observe that at every timestep we have the difference between two quadratic functions of \( z \), so can complete the square to combine this into a single quadratic function. Also note that this term appears in both the numerator and denominator so the constant normalizing (\( Z_2 \)) terms will factor out of the integral and cancel out, so we can remove them from the equation.

\[
\zeta_{t_j} = (z_{t_j} + \Sigma_d \Sigma_1^{-1} \mu_{t_j})^T \Sigma_d^{-1} (z_{t_j} + \Sigma_d \Sigma_1^{-1} \mu_{t_j}) - \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j} - (\Sigma_d \Sigma_1^{-1} \mu_{t_j})^T \Sigma_d^{-1} (\Sigma_d \Sigma_2^{-1} \mu_{t_j})
\]

(15)

Where \( \Sigma_d = (\Sigma_1^{-1} - \Sigma_2^{-1})^{-1} \). Now we expand out the left most term inside the exponent:

\[
\zeta_{t_j} = z_{t_j}^T \Sigma_d^{-1} z_{t_j} + \mu_{t_j}^T \Sigma_1^{-1} \mu_{t_j} + \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j} - \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j} - \mu_{t_j}^T \Sigma_1^{-1} \mu_{t_j} - (\Sigma_d \Sigma_1^{-1} \mu_{t_j})^T \Sigma_d^{-1} (\Sigma_d \Sigma_2^{-1} \mu_{t_j})
\]

(16)

Notice that the two underlined terms are the same, except for the sign, so they cancel out and we’re left with:

\[
\zeta_{t_j} = z_{t_j}^T \Sigma_d^{-1} z_{t_j} + 2\mu_{t_j}^T \Sigma_1^{-1} z_{t_j} - \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j}
\]

(17)

Now define \( \hat{z}_j = z_{t_j} - \mu_{t_j} = \frac{x_{t_j} - x_{t_{j-1}}}{dt} - f(\hat{x}_{t_j}) + G(\hat{x}_{t_j})u_{t_j} \). We re-write this equation in terms of \( \hat{z} \):

\[
\zeta_{t_j} = (\hat{z}_{t_j} + \mu_{t_j})^T \Sigma_1^{-1} (\hat{z}_{t_j} + \mu_{t_j}) + 2\mu_{t_j}^T \Sigma_2^{-1} (\hat{z}_{t_j} + \mu_{t_j}) - \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j}
\]

(18)

which expands out to:

\[
\zeta_{t_j} = \hat{z}_{t_j}^T \Sigma_d^{-1} \hat{z}_{t_j} + 2\mu_{t_j}^T \Sigma_1^{-1} \hat{z}_{t_j} + \mu_{t_j}^T \Sigma_1^{-1} \mu_{t_j} + 2\mu_{t_j}^T \Sigma_2^{-1} \hat{z}_{t_j} + 2\mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j} - \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j}
\]

\[
= \hat{z}_{t_j}^T \Sigma_d^{-1} \hat{z}_{t_j} + 2\mu_{t_j}^T \Sigma_1^{-1} \hat{z}_{t_j} + \mu_{t_j}^T \Sigma_1^{-1} \mu_{t_j} + 2\mu_{t_j}^T \Sigma_2^{-1} \hat{z}_{t_j} + \mu_{t_j}^T \Sigma_1^{-1} \mu_{t_j} - \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j}
\]

Now recall that \( \Sigma_d = (\Sigma_1^{-1} - \Sigma_2^{-1})^{-1} \), so we can split the quadratic terms in \( \Sigma_d^{-1} \) into the \( \Sigma_1^{-1} \) and \( \Sigma_2^{-1} \) components:

\[
\zeta_{t_j} = \hat{z}_{t_j}^T \Sigma_1^{-1} \hat{z}_{t_j} + 2\mu_{t_j}^T \Sigma_1^{-1} \hat{z}_{t_j} + \mu_{t_j}^T \Sigma_1^{-1} \mu_{t_j} - \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j} + 2\mu_{t_j}^T \Sigma_2^{-1} \hat{z}_{t_j} + \mu_{t_j}^T \Sigma_1^{-1} \mu_{t_j} - \mu_{t_j}^T \Sigma_2^{-1} \mu_{t_j}
\]

(19)
and by noting that the underlined terms cancel out we see that we’re left with:

\[ \zeta_t = \bar{z}_{t_j}^T \Sigma_2^{-1} \bar{z}_{t_j} + 2\mu_{t_j}^T \Sigma_1^{-1} \bar{z}_{t_j} + \mu_{t_j}^T \Sigma_1^{-1} \mu_{t_j} \]

The first term in this additional cost can be interpreted as a penalty for over-aggressive sampling, while the second portion is a penalty for shifting the mean. Lastly we define \( \lambda \Gamma = \Sigma_2 \), and recall the identity \( \Sigma_1 = \lambda G_c R^{-1} G_c^T = \lambda H \). Substituting these terms into (14) results in:

\[
\frac{p(\tau_i)}{q(\tau_i)} = \frac{Z_2}{Z_1} \exp \left( -\frac{dt}{2} \sum_{j=i}^N \left( \bar{z}_{t_j}^T (H^{-1} - \Gamma^{-1}) \bar{z}_{t_j} + 2\mu_{t_j}^T H^{-1} \bar{z}_{t_j} + \mu_{t_j}^T H^{-1} \mu_{t_j} \right) \right)
\]

We then fold this term into the state dependent portion of the cost-to-go and define the total cost-to-go \( \hat{S}(\tau_i) \) as:

\[
\hat{S}(\tau_i) = \phi(x_{t_N}) + \sum_{j=i}^N \left( q(x_{t_j}) + \frac{1}{2} \bar{z}_{t_j}^T (H^{-1} - \Gamma^{-1}) \bar{z}_{t_j} + \mu_{t_j}^T H^{-1} \bar{z}_{t_j} + \frac{1}{2} \mu_{t_j}^T H^{-1} \mu_{t_j} \right) dt
\]

And then we can approximate the optimal controls by randomly drawing samples from the distribution \( q(\tau_i) \) whose covariance is defined by the exploration variance \( \Sigma_2 \) and with mean defined by the previously computed controls. The final approximation is:

\[
R^{-1} G_c^T H^{-1} \left( \sum_{k=1}^K \exp \left( -\frac{1}{\lambda} \hat{S}(\tau_i^k) \right) z_{t_i} \right) \rightarrow u^*(t_i)
\]

And if we express this in terms of \( \tilde{z} \) it becomes:

\[
R^{-1} G_c^T H^{-1} \left( \sum_{k=1}^K \exp \left( -\frac{1}{\lambda} \hat{S}(\tau_i^k) \right) (\tilde{z}_{t_i} + G_c u(x_{t_i})) \right)
\]

Lastly we note that \( \tilde{z}_{t_i} \approx B_c \frac{\epsilon}{\sqrt{dt}} \) where \( B_c \) is the Cholesky decomposition of the exploration variance and \( \epsilon \) is a standard normal random variable. Additionally note that in the case where \( G_c \) is square and invertible we have equation (23) reduces to:

\[
u(x_{t_i}) + G^{-1} \left( \sum_{k=1}^K \exp \left( -\frac{1}{\lambda} \hat{S}(\tau_i^k) \right) B_c \frac{\epsilon}{\sqrt{dt}} \right)\]

### 3 Model Predictive Control

Most previous applications of path integral control have focused on either learning the parameters of a pre-specified policy in a model free reinforcement learning setting, or on planning an open loop control sequence while not moving. In contrast we pursue a model predictive control approach where a control sequence is computed, then a small portion of the control is executed, and then the sequence is re-optimized. This loop continues on indefinitely. The benefit of this approach is (1) We are not limited by a potentially sub-optimal control parametrization, (2) The ostensibly open loop control law computed by (24) is converted into an implicit feedback controller, and (3) By using a sliding window approach the controller can operate over arbitrarily long time horizons. This method clearly requires a model of the system, however we since we only require the model for forward simulation it would be possible to use a learned instead of analytic model.

Another drawback, and the main reason why this type of control has not seen widespread application, is that computing (24) fast enough for real-time control is very difficult. In order to get a good approximation to the optimal control, we need to be able to sample a large number of trajectories in real-time. The two existing methods for solving this problem are to either (1) Use a hierarchical scheme to reduce the complexity of the system to one that can be sampled from efficiently [12], or (2) Use a graphics processing unit to sample thousands of trajectories in parallel [13]. Although in certain instances hierarchical control methods work very well, they can be problematic when trying to perform maneuvers which are highly dependent on the system under consideration (since planning
is done without reference to the dynamics). The GPU based approach makes no restrictions on the
dynamics and considers them in their full complexity when computing the control. Therefore we
use the GPU based method in our MPC implementation.

One important detail to note is that the computation of (24) yields an approximation for the optimal
controls for every time-step in the time-horizon. This allows for the optimization to be warm-started
at the next iteration by sliding the whole control sequence down one-timestep after a control is
executed, by the time a control is actually executed it’s undergone at least as many optimization
iterations as the number of steps in the time horizon.

4 Experiments

We tested the model predictive path integral control algorithm (MPPI) on a simulated platform of a
quadrotor attempting to navigate an obstacle filled environment. We used a model predictive version
of the differential dynamic programming (DDP) algorithm as a baseline comparison.

Single Quadrotor: The quadrotor task was to fly through a field filled with cylindrical obstacles as
fast as possible. We randomly generated three forest, one where obstacles are on average 3 meters
apart, the second one 4 meters apart, and the third 5 meters apart. And then separately created
cost functions for both MPPI and DDP which guides the quadrotor through the forest as quickly as
possible.

\[ q(x) = 2.5(p_x - p_{x,des})^2 + 2.5(p_y - p_{y,des})^2 + 150(p_z - p_{z,des})^2 + 50\psi^2 + \|v\|^2 + 350 \exp\left(-\frac{d^2}{12}\right) + 1000C \]

where \((p_x, p_y, p_z)\) denotes the position of the
vehicle. \(\psi\) denotes the yaw angle in radians, \(v\) is velocity, and \(d\) is the distance to the closest
obstacle. \(C\) is a variable which indicates whether the vehicle has crashed into the ground or an
obstacle. Additionally if \(C = 1\) (which indicates a crash), the rollout stops simulating the dynamics
and the vehicle remains where it is for the rest of the time horizon. We found that the crash indicator
term is not useful for the MPC-DDP based controller, as one would expect since the discontinuity
it creates is difficult to approximate with a quadratic function. Therefore the term in the cost for
avoiding obstacles in the MPC-DDP controller consists purely of a much larger exponential term:

\[ 2000 \sum_{i=1}^{N} \exp\left(-\frac{1}{2}d_i^2\right) \]

note that this sum is over all the obstacles in the proximity of the vehicle whereas the MPPI controller
only has to consider the closest obstacle. Since the MPPI controller can explicitly reason about crashing (as opposed to just staying away from obstacles), it’s able to travel both faster and closer to obstacles than the MPC-DDP controller. Figure (1) shows the difference in
time and the trajectories taken by MPC-DDP and one of the MPPI runs on the forest with obstacles
placed on average 4 meters away.

Multiple Quadrotors: In this set of experiments we tested the algorithms ability to simultaneously
control multiple quadrotors operating in close proximity to each other, this was done by combining
several quadrotors into one large system and then attempting the same obstacle navigation task. The
quadrotor was able to successfully control 3 quadrotors at once through the obstacle field in real-
time. Generally, it was also able to control 9 quadrotors performing the same task (although in our
implementation the 9 quadrotor system ran slightly above real-time). We also set the obstacles to
move at a rate of 3 m/s in a random direction. The results for the cases of 1, 3 and 9 quadrotors
are illustrated in figure (3). These results indicate the improvement in the performance of MPPI as the number of the rollouts increases for the case 1 and 3 quad-rotors when moving in environments with static obstacles figure (3b) and environment with moving obstacles figure (3a). For the case of 9 quadrotors the performance improvement is similar. However here we report the percentage of completion since for the case of 9 quadrotors there are situations where the task can not be complete due to crashing of one vehicle in a obstacle. This is illustrated in figure (3c) where the completion percentages is above 90% as the number of the rollouts increases.

5 Discussion

In this work we have presented a novel Model Predictive Path Integral (MPPI) control algorithm based on a generalized importance sampling scheme. This scheme allows for changing the drift and diffusion terms of the sampling distribution. The MPPI scheme operates on the fully nonlinear dynamics of the system under consideration without splitting the problem into planning and control. This allows for aggressive maneuvering and full exploitation of the nonlinear dynamics. We have compared the proposed algorithm with Model Predictive Control formulations of DDP on a navigation task with a quadrotor. In addition we have applied MPPI on navigation tasks for a team of 3 and 9 quad rotors. Our results demonstrate the outperformance of MPPI against MPC-DDP for systems with highly non-linear dynamics or non-smooth costs, which are typical for autonomous systems.

Acknowledgment

This research has been supported by NSF Grant No. NRI-1426945.

References


