Implicit Sampling for Path Integral Control, Monte Carlo Localization, and SLAM

Implicit sampling is a recently developed variationally enhanced sampling method that guides its samples to regions of high probability, so that each sample carries information. Implicit sampling may thus improve the performance of algorithms that rely on Monte Carlo (MC) methods. Here the applicability and usefulness of implicit sampling for improving the performance of MC methods in estimation and control is explored, and implicit sampling based algorithms for stochastic optimal control, stochastic localization, and simultaneous localization and mapping (SLAM) are presented. The algorithms are tested in numerical experiments where it is found that fewer samples are required if implicit sampling is used, and that the overall runtimes of the algorithms are reduced.

[DOI: 10.1115/1.4029064]

1 Introduction

Many problems in physics and engineering require that one produces samples of a random variable with a given probability density function (PDF) \( p \) [1,2]. If \( p \) is difficult to sample, one can sample an easier-to-sample PDF \( p_0 \) (the importance function), and attach to each sample \( X_j \sim p_0 \), \( j = 1, \ldots, M \), the weight \( W_j = p(X_j) / p_0(X_j) \) (capital letters denote realizations of random variables, and these realizations are indexed in superscript; subscript indices label discrete time). The weighted samples \( X_j \) form an empirical estimate of the PDF \( p \) and, under mild assumptions, the empirical estimate converges weakly to \( p \), i.e., expected values of a function \( h \) with respect to \( p \) can be approximated by weighted averages of the function values \( h(X_j) \). The difficulty is to find a “good” importance function \( p_0 \); if \( p_0 \) is not a good approximation of \( p \), for example, if \( p_0 \) is not large when \( p \) is large, then many of the samples one proposes using \( p_0 \) may have a small probability with respect to \( p \) and, therefore, carry little information, and make sampling inefficient.

Implicit sampling is a general method for constructing useful importance functions [3,4]. The basic idea is to first locate the region of high probability with respect to \( p \) via numerical optimization, and then to map a reference variable to this region; this mapping is done by solving algebraic equations. While the cost of generating a sample with implicit sampling is larger than in many other MC schemes, implicit sampling can be efficient because the samples carry more information so that fewer samples are required. In Refs. [5–7], this tradeoff was examined in the context of geophysics, where it was found that implicit sampling indeed can be efficient, in particular when the dimension of the problem is large.

Here, implicit sampling is applied to estimation and control in robotics, and new algorithms for stochastic optimal control, Monte Carlo localization (MCL), and SLAM are developed. Implementations and efficiencies of these algorithms are illustrated and explored with examples. In particular, it is investigated if implicit sampling, which requires fewer samples that are, however, more informative and more expensive, can be efficient compared to other sampling schemes that may require more samples, each of which is less informative, however cheaper to generate.

The optimal control of a stochastic control problem can be found by solving a stochastic Hamilton–Jacobi–Bellman (HJB) equation [8]. The dimension of the domain of this partial differential equation (PDE) equals the dimension of the state space of the control problem. Classical PDE solvers require a grid on the domain and, therefore, are impractical for control problems of moderate or large dimension. For a class of stochastic control problems, one can use MC solvers instead of grid based PDE techniques and, since the MC approach avoids grids, it is in principle feasible to solve larger dimensional control problems within this class [9–12]. However, the sampling scheme must be chosen carefully or else MC based PDE solving will also fail (in the sense that unfeasibly many samples are required). It is shown in Sec. 3 how to apply implicit sampling to obtain a practical algorithm for stochastic optimal control that avoids many of the pitfalls one faces in MC based PDE solving. The method and algorithm are illustrated with the double slit problem (see Sec. 3.1 and Ref. [9]), which is a simple but vivid example of how things can go wrong, and how they can be fixed with implicit sampling. Using an inverse dynamics controller for a two degrees-of-freedom robotic arm as an example, it is also shown how to obtain a semi-analytic solution for a linear Gaussian problem via implicit sampling. Finally, it is indicated how implicit sampling and stochastic optimal control can help with being trapped in local minima of non-convex optimization problems. An extension of the method presented here is also discussed in the conference paper [13].

In robotics, one often updates the predictions of a dynamic model of an autonomous robot with the output of the robot’s sensors (e.g., radar or lidar scans). This problem is often called “localization,” and can be formulated as a sampling problem. Localization algorithms that rely on MC sampling for the computations are called MCL [14]. One can also learn the geometry and configuration of the map while simultaneously localizing the robot in it, which leads to the problem of “SLAM.” Efficient numerical solutions of the MCL and SLAM problems are a fundamental requirement for autonomous robots [15,16]. The various MCL and SLAM algorithms differ in their importance function \( p_0 \), and some algorithms have been shown to be inefficient due to a poor choice of \( p_0 \) [15]. Here it is shown how to use implicit sampling to generate an efficient importance function for MCL and SLAM. The implicit sampling based MCL and SLAM algorithms are convergent, i.e., as the number of samples goes to infinity one obtains an empirical estimate of the true posterior even if the underlying dynamics are nonlinear (whereas many other SLAM algorithms require linearity for convergence [15]). The memory requirements of the implicit sampling based SLAM algorithm scale linearly with the number of features in the map. The efficiencies of the new MCL and SLAM algorithms are compared to the efficiencies...
of competing MCL and SLAM algorithms in numerical experiments with the data set [17].

2 Review of Implicit Sampling

The efficiency of MC sampling depends on how well the importance function \( p_0 \) approximates the target PDF \( p \). In particular, \( p_0 \) must be large where \( p \) is large, or the samples one produces using \( p_0 \) are unlikely with respect to \( p \). Implicit sampling is a general MC sampling technique that constructs an importance function that is large where \( p \) is large by mapping a reference variable to the region where \( p \) is large [3–7]. Here implicit sampling is described in general terms; below implementations of implicit sampling are described in the context of applications.

The region where \( p \) is large is the region where \(- \log(p)\) is small (the logarithm is used here because in applications \( p \) often involves exponential functions). The region where \( p \) is large can thus be located via minimization of the function

\[
F(x) = -\log(p(x))
\]

If \( F \) is convex, the minimizer \( \mu = \arg \min F \) (i.e., the mode of \( p \)) is the most likely value, and high-probability samples are in its neighborhood. One can obtain samples in this region by solving the stochastic algebraic equation

\[
F(x) - \phi = \frac{1}{2} \xi^T \xi
\]

where \( \phi = \min F, \xi \sim \mathcal{N}(0, I_m) \) is an \( m \)-dimensional Gaussian reference variable and where \( T \) denotes a transpose and \( I_m \) is the identity matrix of order \( m \); here, and for the remainder of this paper, \( \mathcal{N}(a, B) \) is short-hand notation for a Gaussian PDF with mean \( a \) and covariance matrix \( B \). Note that the right-hand side of Eq. (1) is likely to be small because \( \xi \) is close to the origin, which implies that the left-hand side is small, which in turn means that the solution of Eq. (1), i.e., the sample, is close to the mode and, thus, in the high-probability region.

The weights of the samples are proportional to the absolute value of the Jacobian determinant of the map that connects the sample \( X \) to the reference variable \( \xi \)

\[
W(X) \propto \left| \det \left( \frac{\partial x}{\partial \xi} \right) \right|
\]

In practice, the weights are usually normalized so that their sum is one. Various ways of constructing a map from \( x \) to \( \xi \) have been reported in Refs. [4,7] two of which are summarized below. In summary, implicit sampling requires: (1) minimizing \( F = -\log(p(x)) \) and (2) solving Eq. (1). This two-step approach makes efficient use of the available computational resources: the minimization identifies the high probability region and the samples are focused to lie in this region, so that (almost) all samples carry information. This is in contrast to other sampling schemes where an importance function is chosen ad hoc, which often means that many of the samples carry little or no information; the computations used for generating uninformative samples are wasted.

Finally, the assumption that \( F \) is convex can be relaxed. Implicit sampling can be used without modification if \( F \) is merely \( U \)-shaped, i.e., the target PDF, \( p \), is unimodal. Multimodal target PDFs can be sampled, e.g., by using mixture models, for which one approximates each mode using the above recipe (see Refs. [4,7,18] for more details).

2.1 Generating Samples With Random Maps.

To generate samples, one solves Eq. (1) with a one-to-one and onto mapping. There are many choices for such a mapping, and one is to solve Eq. (1) in a random direction, i.e., one puts

\[
X^I = \mu + \xi L^{-T} \eta^I
\]

where \( L \) is a Cholesky factor of the Hessian of \( F \) at the minimum, \( \eta^I = \xi / \sqrt{\xi^T \xi} \) is a vector which is uniformly distributed on the \( m \)-dimensional hypersphere, and where \( \xi \) is a scalar. One then computes \( \xi \) by substituting Eq. (2) into Eq. (1). This approach, called “random map,” requires solving a scalar equation to generate a sample [7]. Moreover, the Jacobian of this map is easy to evaluate with the formula (see Ref. [7] for a derivation)

\[
\det \left( \frac{\partial x}{\partial \xi} \right) = \frac{\left( \xi^T \xi \right)^{(1-m)/2}}{\det(L)} \frac{\xi(\xi)^{m-1}}{2\sqrt{\det(L)}}
\]

where \( \nabla F \) is the gradient of \( F \) with respect to \( x \) (an \( m \)-dimensional row vector).

In numerical implementations of this method, calculating \( \nabla F \) may require repeated evaluations of \( F \), e.g., if the gradient is approximated via finite differences. This can be avoided by noting that

\[
\frac{d\xi}{dp} = \frac{1}{2\sqrt{\det(L)}}
\]

where \( \rho = \xi^T \xi \), so that the Jacobian becomes

\[
\det \left( \frac{\partial x}{\partial \xi} \right) = \rho^{(1-m)/2} \frac{\xi(\xi)^{m-1} \frac{d\xi}{dp}}{\det(L)}
\]

The scalar derivative \( d\xi/\rho \) can be evaluated using finite differences with a few evaluations of \( F \) (the precise number of function evaluations depends on the finite difference scheme one chooses).

2.2 Generating Samples With Approximate Quadratic Equations.

Another path to solving Eq. (1) is to replace it with an approximate quadratic equation

\[
\hat{F}(x) - \phi = \frac{1}{2} \xi^T \xi
\]

where

\[
\hat{F}(x) = \phi + \frac{1}{2} (x - \mu)^T H (x - \mu)
\]

is the Taylor expansion of order two of \( F \) about its minimizer \( \mu \), and where \( H \) is the Hessian of \( F \) at the minimum. A solution of Eq. (5) is

\[
X^I = \mu + L^{-T} \hat{\xi}_j
\]

where \( L \) is the Cholesky factor of \( H = LL^T \). The weights are

\[
W^I \propto \exp(-\phi + \hat{F}(X^I) - F(X^I))
\]

and account for the error one makes by solving Eq. (5) instead of Eq. (1) (see Ref. [4] for more details).

3 Application to Path Integral Control

The finite horizon stochastic optimal control problem considered here is as follows: find a control \( u \) (a \( p \) dimensional vector function of the state \( x \)) that minimizes the cost function

\[
C(x_0, t_0, u) = \mathbb{E} \left[ \Phi(x(t_f)) + \int_{t_0}^{t_f} u(x(\tau), \tau)^T R u(x(\tau), \tau) + V(x(\tau), \tau) d\tau \right]
\]

where \( x \) is an \( m \)-dimensional vector (the state), \( t_0 \leq t \leq t_f \) is time, \( t_f \) is the final time (or the horizon), \( \Phi \) is a scalar function that
describes the “final cost,” $R$ is a $p \times p$ symmetric positive definite matrix that specifies the control cost, and $V$ is a scalar function which describes the “running cost” (which is also called “potential”); the equation is taken over trajectories of the stochastic differential equation (SDE)

$$dx = f(x,t)dt + G(udt + QdW)$$  \hspace{1cm} (7)

starting at $x(t_0) = x_0$, where $f$ is a smooth $m$-dimensional vector function which describes the dynamics, and where $G$ and $Q$ are $m \times p$ and $p \times r$ matrices that describe how the uncertainty and controls are distributed within the system; $W$ is Brownian motion (see, e.g., Ref. [8] for more details about stochastic optimal control; I closely follow Ref. [9] in the presentation of path integral control).

The “optimal cost-to-go” is defined as

$$J(x,t) = \min_u C(x,t,u)$$

and satisfies the stochastic HJB equation [8]

$$\partial_t J(x,t) = \partial_u J(x,t) + \text{Tr}(Q^T G^T \partial_u J(x,t) G) + V(x,t) - \frac{1}{2} \partial_u J(x,t) G G^T R^{-1} G \partial_u J(x,t)$$

where $\text{Tr}$ is the trace. If there exists a scalar $\gamma$ such that

$$\gamma G R^{-1} G^T = G Q Q^T G^T$$  \hspace{1cm} (8)

then the nonlinear change of variables

$$J(x,t) = -\gamma \log \psi(x,t)$$

linearizes the stochastic HJB equation and one obtains

$$\partial_t \psi = \frac{V(x,t)}{\gamma} \psi - f(x,t)^T \partial_x \psi - \frac{1}{2} \text{Tr}(Q^T G^T \partial_x \psi G Q)$$  \hspace{1cm} (9)

with final condition $\psi(x,t_f) = \exp(-\Phi(x(t_f))/\gamma)$ and with the optimal control

$$u = -R^{-1} \partial_x JG$$  \hspace{1cm} (10)

see Refs. [9,19]. Thus, one can compute the optimal control by solving the HJB equation. Numerical PDE solvers typically require a grid on the domain of the PDE; however, the domain has the dimension of the state space of the control problem. Thus, grid based numerical PDE techniques only apply to control problems of a low dimension (or else the computational requirements become excessive).

For larger dimensional problems, one can use stochastic PDE solvers which do not require a grid. In particular, one can solve the adjoint equation

$$\partial_t \psi = -\frac{V(x,t)}{\gamma} \psi - \partial_x (f(x,t) \psi) + \frac{1}{2} \text{Tr}(Q G \partial_x \psi G^T Q^T)$$

forward in time (instead of solving Eq. (9) backward in time) with the Feynman–Kac formula (see, e.g., Ref. [11])

$$\psi(x,t) = E \left[ \exp \left( -\frac{1}{2} \Phi(y(t_1), t_1) + \int_{t_1}^t V(y(\tau), \tau) d\tau \right) \right]$$  \hspace{1cm} (11)

where the expectation is over the trajectories of

$$dy = f(y, \tau) d\tau + G dW$$  \hspace{1cm} (12)

starting at $y(\tau = t) = x(t)$. This is the path integral formulation of stochastic optimal control [9–12]. Note that the class of problems that can be tackled with path integral control is rather general, since the assumptions of: (1) a quadratic control cost and (2) the condition on the noise and the control cost in Eq. (8) are not restrictive. In particular, the dynamics $f(x,t)$ and the potential $V(x,t)$ in Eq. (7) can be nonlinear.

To evaluate the Feynman–Kac formula numerically, one approximates the infinite dimensional integral in Eq. (11) by a finite dimensional one. For example, one can discretize the integral on a regular grid in time with constant time step $\Delta t$, so that

$$\psi(x,t) \approx \sum_{y_1, \ldots, y_n} p(y_1, \ldots, y_n) \exp(-\Phi(y_1, \Delta t)/\gamma)$$

$$\times \exp \left( -\frac{1}{2} \Phi(y_n, n \Delta t) + \frac{\Delta t}{2\gamma} \sum_{i=1}^n \left( \frac{\Delta y_i}{\Delta t} - f(y_i, \tau_i) \right)^T \Sigma^{-1} \left( \frac{\Delta y_i}{\Delta t} - f(y_i, \tau_i) \right) \right)$$  \hspace{1cm} (13)

where $y_1 = x(t)$, $n = (t_f - t)/\Delta t$, $\tau_i = t + i \Delta t$, $i = 0, 1, \ldots, n$, and where $p$ is the PDF of the discretized trajectory $y_1, \ldots, y_n$. Here the trapezoidal rule is used to discretize the integral of the potential; however, other choices are also possible [20]. The SDE (12) implies that the increments $\Delta y_i = y_i - y_{i-1}$ are independent Gaussians, e.g., for a forward Euler discretization [21] of Eq. (12)

$$\Delta y_i \sim N(f(y_i, \tau_i) \Delta t, \Sigma \Delta t)$$

where $\Sigma = \gamma G R^{-1} G^T = G Q Q^T G^T$, so that

$$p(y_1, \ldots, y_n) = \prod_{i=1}^n p(\Delta y_i) = \frac{\exp \left( -\frac{1}{2} \sum_{i=1}^n \frac{\Delta y_i}{\Delta t} - f(y_i, \tau_i) \right)^T \Sigma^{-1} \left( \frac{\Delta y_i}{\Delta t} - f(y_i, \tau_i) \right)}{(2\pi \Delta t \det \Sigma)^{n/2}}$$  \hspace{1cm} (14)

Thus,

$$\psi(x,t) \approx \frac{1}{(2\pi \Delta t \det \Sigma)^{n/2}} \int dy_1 \ldots dy_n \exp(-F(y_1, \ldots, y_n))$$

where

$$F(y_1, \ldots, y_n) = \frac{1}{\gamma} \Phi(y_n, n \Delta t) + \frac{\Delta t}{2\gamma} \sum_{i=1}^n V(y_i, \tau_i) + V(y_{i-1}, \tau_{i-1}) + \frac{1}{2} \sum_{i=1}^n \Delta t \left( \frac{\Delta y_i}{\Delta t} - f(y_i, \tau_i) \right)^T \Sigma^{-1} \left( \frac{\Delta y_i}{\Delta t} - f(y_i, \tau_i) \right)$$  \hspace{1cm} (15)
Note that this $F$ depends on how one discretizes the integral of the potential $V$, and the SDE (12); other discretization schemes will lead to different $F$s.

For a given discretization, i.e., for a given $F$, MC sampling can be applied to compute the expectation in Eq. (13). For example, one can evaluate

$$G(y_1, \ldots, y_n) = \frac{1}{J} \Phi(y_n, n\Delta t) + \frac{\Delta t}{2J} \sum_{i=1}^{J} V(y_i, \tau_i) + V(y_{i-1}, \tau_{i-1})$$

for discretized trajectories of Eq. (12), followed by averaging. However, this method can fail if the potential $V$ has deep wells [9–11]. In this case, many trajectories of Eq. (12) can end up where $V$ is large and, thus, contribute little to the approximation of the expected value $\psi$. For efficient sampling, one needs a method that guides the samples to remain where the potential is small. This guiding of the samples can be achieved via implicit sampling. Recall that in implicit sampling one first locates the region of high probability by minimizing $F = -\log(p(x))$, where $p(x)$ is the PDF of the random variable one is interested in. The trajectories we wish to sample have the joint PDF (14), so that, in order to apply implicit sampling to path integral control, one needs to minimize $F$ in Eq. (15). With this $F$, one solves the algebraic equation (1) with a one-to-one and onto map (2). The integral in Eq. (13) becomes

$$\psi(x, t) \approx \frac{\exp(-\phi)}{(2\pi\Delta t \det \Sigma)^{n/2}} \int d\xi_1 \cdots d\xi_n \exp(-\frac{1}{2} \Sigma^{-1} \xi) \frac{\partial}{\partial \xi}$$

where the expectation is over the reference variable $\xi$ and where $\partial \xi / \partial \xi$ is the Jacobian of the map from $t$ to $\xi$ in Eq. (2). Combining Eq. (16) with the expression (3) for the Jacobian gives

$$\psi(x, t) \approx \frac{\exp(-\phi)}{(2\Delta t \det \Sigma)^{n/2} \det(L)} E\left[ \left( \frac{1}{2} \frac{\partial}{\partial \xi} \right) \left( \frac{1}{2} \frac{\partial}{\partial \xi} \right) \right]$$

The expected value is now straightforward to compute via MC, i.e., sampling a the reference variable $\xi$ to obtain $M$ samples $\xi^j, j = 1, \ldots, M$, computing, for each one, $\frac{\partial}{\partial \xi}$ and the gradient of $F$, followed by averaging. In numerical implementations, it may be more efficient to use the equivalent expression (3) for the Jacobian (which avoids computing the gradient of $F$). In this case, one obtains

$$\psi(x(t), t) \approx \frac{\exp(-\phi)}{(2\Delta t \det \Sigma)^{n/2} \det(L)} E\left[ \left( \frac{1}{2} \frac{\partial}{\partial \xi} \right) \left( \frac{1}{2} \frac{\partial}{\partial \xi} \right) \right]$$

Once $\psi(x, t)$ is computed, we can compute the optimal control from Eq. (10), e.g., via finite differences. It is important to note that this defines the optimal control at time $t$, and that the computations have to be repeated at the next time step to compute $\psi(x(t + dt), t + dt)$. The use of implicit sampling in stochastic optimal control is illustrated with three examples.

3.1 The Double Slit Problem. The double slit problem is a simple example that illustrates the pitfalls one must avoid when using MC sampling for path integral control [9]. The problem is as follows. Suppose you observe a somehow confused person walking (randomly) toward a wall with two doors, and your job is to guide this person through either one of the two doors. The person and you are modeled by the controlled dynamics

$$dx = u dt + \sqrt{\sigma} dW$$

the final cost is quadratic, $\Phi = x(t_f)^2/2$, and the scalar $R > 0$, that defines the cost of the control, is given; the “wall” is modeled by the potential

$$V(x, t) = \begin{cases} \infty & \text{if } t = t_f \text{ and } x < a, \text{ or } b < x < c, \text{ or } d < x \\ 0 & \text{otherwise} \end{cases}$$

for given $a < b < c < d$ and $t_f > 0$. The stochastic HJB equation of the double slit problem is

$$\psi(x, t) = -\frac{Q}{\gamma} + \frac{1}{2} \sigma^2 \psi_{xx}$$

where $\gamma = r \sqrt{\sigma}$ satisfies Eq. (8).

One can attempt to solve this equation using standard MC sampling as follows. Solve the SDE

$$dy = \sqrt{\sigma} dW$$

starting at $y(t) = x(t)$ repeatedly, e.g., using a uniform grid in time and the forward Euler scheme [21]

$$y_{n+1} = y_n + \sqrt{\sigma} \Delta t w_n$$

where $\Delta w_n$ are independent Gaussians with mean 0 and variance 1. For each trajectory $\{y_1, \ldots, y_n\}$, evaluate

$$G(y_1, \ldots, y_n) = \frac{1}{2} \| y_n^2 + \frac{\Delta t}{2} \sum_{i=1}^n V(y_i, \tau_i) + V(y_{i-1}, \tau_{i-1}) \text{ for } a < b < c < d$$

With a high probability, the trajectories will hit the potential wall at $t_f$ and, thus, $G$ is infinite, so that the contribution of a trajectory to the expected value $\psi(x, t)$ in Eq. (11) is zero with a high probability. The situation is illustrated with a simulation using the parameters of Table 1. The results are shown in the left panel of Fig. 1. One observes that 5000 unguided random walks from Eq. (17), all starting at $x(0)$, hit the potential wall, and, thus, score $G = \infty$. All 5000 samples thus carry no probability and do not contribute to the approximation of $\psi$. Even when $O(10^3)$ samples are considered, it is unlikely that sufficiently many make it past the potential wall [9]. We conclude that this method is unfeasible for this problem, since the number of samples required is extremely large due to the deep wells in the potential.

Implicit sampling can be applied here to fix these problems. In particular, one finds from the Feynman–Kac formula that

$$\psi(x, t) = E\left[ \left( \frac{y(t_f)^2}{2} - \frac{1}{2} \int_0^{t_f} V(y(t), \tau) d\tau \right) \right]$$

which one can compute with implicit sampling in two parts. For $t \geq t_f$, i.e., after one has passed the potential wall, the probability of each path is Gaussian with variance $s = t_f - t$, so that

$$\psi(x, t) = \frac{1}{\sqrt{2\pi s}} \exp\left( -\frac{(x - y)^2}{2s} \right)$$

For implicit sampling, one defines

$$F(y) = \frac{(x - y)^2}{2s} + \frac{z^2}{2\gamma}$$

whose minimizer and minimum are $\mu = \arg\min F = y_t / (s + \gamma)$, $\phi = \min F = \phi / (s + \gamma)$. Since $F$ is quadratic, $F = \phi + H(y - \mu^2)/2$.

<table>
<thead>
<tr>
<th>Table 1 Parameters of the double slit problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>Final time</td>
</tr>
<tr>
<td>Critical time</td>
</tr>
<tr>
<td>Slits</td>
</tr>
<tr>
<td>Initial position</td>
</tr>
<tr>
<td>Variance of the noise</td>
</tr>
<tr>
<td>Control cost</td>
</tr>
<tr>
<td>Time step</td>
</tr>
</tbody>
</table>

Transactions of the ASME
Implicit sampling uses the information about the potential which is quadratic because it locates the wells in the potential.

The above analytic solution is compared to a numerical implementation of implicit sampling, for which the paths are discretized with a constant time step $\Delta t$. Here the numerical integration is also split up into two parts. For $t < t_1$, the function $F$ is quadratic because the probabilities are Gaussian and the potential has no effect. Thus,

$$ F = \frac{y^2}{2\gamma} + \frac{1}{2\gamma\Delta t} \sum_{i=1}^n (y_i - y_{i-1})^2 $$

where $n = (t_1 - t)/\Delta t$ and $y_i = y(t + i\Delta t)$, $i = 1, \ldots, n$. Minimizing $F$ is straightforward, and the algebraic equation (1) can be solved with a linear map

$$ y = \mu + L^{-T} \zeta $$

where $y = (y_0, \ldots, y_n)$ is an $n$-dimensional vector, $\mu$ is the minimizer of $F$, and $\zeta$ is an $n$-dimensional vector whose elements are independent standard normal variates; $L$ is a Cholesky factor of the Hessian at the minimum. Since the Jacobian of this linear map is constant, Eq. (16) gives

$$ \psi(x, t) = \frac{\exp(-\phi)}{(y\Delta t)^{n/2} \det L} $$

where $\phi$ is the minimum of $F$.

For $t < t_1$, one obtains the same $F$, but needs to perform a constraint minimization over the slits. There are two (local) minima, one per slit, and both can be found easily using quadratic programming [22,23]. One can then generate a sample close to each of the minima using

$$ y = \mu + L^{-T} \zeta $$

where $y$ again is a vector whose elements are the discretized path and where $\mu$ is the location of a local minimum of the constraint problem and $L$ is a Cholesky factor of the unconstrained Hessian at a minimum. Equation (16) becomes

$$ \psi(x, t) = \frac{\exp(-\phi)}{L^T \zeta} \int_{\text{Slits}} \frac{1}{\det L} d\zeta $$

The expected value of the Jacobian ($1/\det L$) over the slits can be computed by MC as follows. Generate $M$ samples $\zeta_j$, $j = 1, \ldots, M$, and, for each one, compute the corresponding trajectory and check if it hits the potential wall. The integral is $1/\det(L)$ times the ratio of the number of trajectories that pass through the wall and $M$.

The right panel of Fig. 1 shows 50 trajectories one obtains with this approximation at $t = \Delta t$. Note that most of the trajectories pass through the slit, i.e., most of the samples carry a significant probability, score a small $F$, and, thus, contribute to the approximation of the expected value $\bar{\psi}(x, t)$. These “guiding” effects make it
possible to solve the problem with $O(10)$ samples, while the standard MC scheme fails even with $O(10^5)$ samples [9]. The Laplace guided strategy presented in Ref. [9] (which uses similar ingredients as implicit sampling and is also related to the optimal nudging constructions in Refs. [24,25]) requires about 100 samples.

The numerical approximation obtained with implicit sampling (50 samples) is compared to the analytical solution in Fig. 2, where the optimal control and the trajectory under this control are shown. One observes that the numerical approximation of the optimal control is quite close to the control, one obtains from the analytical solution (right panel of Fig. 2) and, hence, the controlled trajectory one obtains with implicit sampling is also close to the one computed from the analytical solution (left panel of Fig. 2). This statement can be made more precise by solving the control problem repeatedly (each solution is a random event) and averaging.

The double slit problem is solved 500 times and the error of the numerical approximation is recorded for each run. The Euclidean norm of the difference of the analytical solution and the approximation via implicit sampling is used to measure an error, in particular in $x$ (the trajectory under optimal control) and $u$ (the optimal control). The number of samples of implicit sampling is varied to study the convergence of the algorithm. The results are shown in Table 2 where the mean and standard deviation of the Euclidean norm of the error in $x$ and $u$, respectively, are listed. These numbers indicate the errors one should expect in a typical run. The errors do not dramatically decrease for larger sample numbers, which indicates that the algorithm has converged. The small variance of the errors indicates that similar errors occur in each run, so that one concludes that implicit sampling is reliable.

The error in the numerical implementation of implicit sampling is mostly due to neglecting one of the local minima of $F$. In numerical tests, only a slight improvement was observed when both minima were used for sampling; however, the computations are twice as fast if one considers only the smaller of the two minima.

### 3.2 Stochastic Control of a Two Degrees-of-Freedom Robotic Arm

Stochastic optimal control of the two degrees-of-freedom robotic arm shown in Fig. 3 is considered. The controller is an “inverse dynamics controller” that linearizes the system. This example demonstrates how implicit sampling based path integral control works in linear problems. Specifically, a semi-analytical solution is derived for which a numerical optimization

### Table 2 Mean errors of the numerical implementation of implicit sampling

<table>
<thead>
<tr>
<th>Number of samples</th>
<th>Error in $x$ in % (mean ± std. dev.)</th>
<th>Error in $u$ in % (mean ± std. dev.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.64 ± 0.63</td>
<td>4.81 ± 2.15</td>
</tr>
<tr>
<td>50</td>
<td>2.70 ± 0.63</td>
<td>4.61 ± 1.89</td>
</tr>
<tr>
<td>100</td>
<td>2.66 ± 0.63</td>
<td>4.54 ± 2.37</td>
</tr>
<tr>
<td>500</td>
<td>2.61 ± 0.62</td>
<td>4.46 ± 2.17</td>
</tr>
</tbody>
</table>
is required; however, the expected value of the implicit sampling solution in Eq. (16) can be evaluated explicitly (i.e., no numerical sampling is needed). Moreover, only some of the dynamic equations of the first-order formulation of the dynamics are driven by noise, so that the dimension of the stochastic subspace is less than the actual dimension of the problem. Implicit sampling can exploit these features, and this example demonstrates how. The algorithm is tested in numerical experiments in which false parameters are given to the algorithm to demonstrate the robustness of the path integral controller to model error.

The goal is to compute two independent torques \( \tau_1 \) and \( \tau_2 \) that drive the arm to a desired position \( \theta^* \) (described by the two-angles \( \theta_1, \theta_2 \), the degrees-of-freedom). Neglecting energy dissipation, and assuming the robot is mounted on a horizontal table (no effects of gravity), the equation of motion is

\[
M(\theta) \ddot{\theta} + C(\theta, \dot{\theta}) \theta = \tau
\]

where \( \theta = (\theta_1, \theta_2) \) is a two-dimensional vector and where dots denote derivatives with respect to time; the \( 2 \times 2 \) matrices

\[
M(\theta) = \begin{pmatrix} a_1 + a_2 \cos(\theta_2) & a_2 + a_3 \cos(\theta_2) \\ a_2 + a_3 \cos(\theta_2) & a_2 \end{pmatrix}
\]

\[
C(\theta, \dot{\theta}) = \begin{pmatrix} -a_1 \sin(\theta_2) \dot{\theta}_2 & -a_3(\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_2) \\ a_3 \sin(\theta_2) \dot{\theta}_1 & 0 \end{pmatrix}
\]

where

\[
a_1 = m_1 l_1^2 + m_2 l_1^2 + m_2 l_2^2 + I_1 + I_2
\]

\[
a_2 = m_2 l_2^2 + I_2
\]

\[
a_3 = I_1 m_2 l_2
\]

define the dynamics and depend on the lengths of the arms \( l_1, l_2 \) and the loads \( m_1, m_2 \) (see, e.g., Ref. [26] for more details about this model and Table 3 for the parameters used in simulations). One can compute the torques by inverting the dynamics and choosing \( u = C(\theta, \dot{\theta}) + M(\theta)u \) to derive the linear system

\[
\begin{pmatrix} \dot{\theta} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \theta + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u
\]

where \( 0 \) denotes the matrix whose elements are all zero (of appropriate dimensions), and \( I \) is the \( 2 \times 2 \) identity matrix; \( u \) is a two-dimensional vector of controls which must be computed. To make the control robust to modeling errors, one can add noise and solve the stochastic optimal control problem

\[
dx = Adt + Gudt + GQdW
\]

where \( x = (\theta, \dot{\theta}) \) is a four-dimensional vector and where the matrices \( A, G, Q \) can be read from Eq. (17), and where \( W \) is a two-dimensional Brownian motion. The final cost is chosen as

\[
\Phi = \| \theta(t_f) - \theta^* \|^2 / 2 + \| \dot{\theta}(t_f) \|^2 / 2
\]

so that the robotic arm stops at \( t_f > 0 \) at the desired position \( \theta^* \). This linear control problem can be solved with linear quadratic Gaussian control [8]; however, here a semi-analytical solution is obtained via implicit sampling.

As before, the time is discretized using a constant time step \( \Delta t = (t_f - t)/n \). The discretized dynamics are

\[
z_{i+1} = z_i + y_i \Delta t
\]

\[
y_{i+1} = y_i + \Delta B
\]

where \( y_i, z_i \) are two-dimensional vectors whose elements are the discretized angular velocities \( \dot{\theta} \) and the discretized angles \( \theta \); note that only one of the above equations is driven by noise \( \Delta B \), since there is no reason to inject noise into the (trivial) equation \( \dot{\theta} = \dot{\theta} \).

The function \( F \) in implicit sampling is thus a function of \( y \) only

\[
F = \frac{1}{2} y^T P y + \frac{1}{2} \left( z - \theta \right)^T Q (z - \theta) + \frac{1}{2} \Delta \sum_{i=1}^{n} \left( y_i - y_{i-1} \right)^2
\]

where \( \gamma = r \) is chosen to satisfy Eq. (8). Because \( F \) is quadratic the minimization is straightforward, and the algebraic equation (1) can be solved with a linear map whose Jacobian is constant and given by the determinant of a Cholesky factor \( L \) of the Hessian of \( F \) at the minimum (see above). Thus, Eq. (16) becomes

\[
\psi(x, t) = \exp \left( -\frac{(\dot{\theta})^T \dot{\theta}}{(\dot{\theta})^T \dot{\theta}} \right)
\]

There is no need for generating samples, since the expected value is computed explicitly (i.e., by evaluating the integral analytically).

The robustness of the implicit sampling based path integral control is tested in numerical experiments. To simulate that the “real” robotic arm behaves differently from the numerical model that the path integral controller uses to find a control, one can give the controller false information about the parameters of the numerical model. Here the parameters \( m_1 \) and \( m_2 \) are perturbed values of the true parameters simulating that the robotic arm picked up a payload (of unknown weight). Thus, the controller works with values \( m_1 = 1.4, m_2 = 1 \); whereas the “true” robotic arm has parameters as in Table 3. The results of a simulation are shown in Fig. 4 where state trajectories and controls are shown for a truly optimal controller (i.e., one who has access to the exact model parameters), and for the path integral controller which uses false model parameters. One observes that the path integral controller can perform the desired task (moving the arm to a new position) and that its control and the resulting state trajectories closely follow those that are generated by a truly optimal controller. This example thus indicates that the path integral controller can perform reliably and quickly in applications where model error may be an issue.

### 3.3 Optimization Via Stochastic Control

One can setup a conventional optimization problem, i.e., find \( \min f(x) \) for a given smooth function \( f \), as a stochastic optimal control problem as follows: find a control \( u \) to minimize the cost function

\[
C(x, u) = E \left[ f(x(t_f)) + \int_0^t u(x, t)^T Ru(x, t)dt \right]
\]

where the expectation is over trajectories of the SDE

\[
dx = u dt + \sqrt{\sigma} dW
\]

The idea is to make use of the stochastic component to explore valleys in the function \( f \). The parameters one can tune are the noise level \( \sigma \), the final time \( t_f \), and the control cost \( R \).

### Table 3 Parameters of the robotic arm

<table>
<thead>
<tr>
<th>Description</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of link one</td>
<td>( l_1 )</td>
<td>1 m</td>
</tr>
<tr>
<td>Distance to the center of mass of link one</td>
<td>( l_1 )</td>
<td>0.5 m</td>
</tr>
<tr>
<td>Distance to the center of mass of link two</td>
<td>( l_2 )</td>
<td>0.5 m</td>
</tr>
<tr>
<td>Mass of link one</td>
<td>( m_1 )</td>
<td>1 kg</td>
</tr>
<tr>
<td>Mass of link two</td>
<td>( m_2 )</td>
<td>1 kg</td>
</tr>
<tr>
<td>Moment of inertia of link one</td>
<td>( l_1 )</td>
<td>2 kg/m²</td>
</tr>
<tr>
<td>Moment of inertia of link two</td>
<td>( l_2 )</td>
<td>2 kg/m²</td>
</tr>
</tbody>
</table>
Implicit sampling for this control problem requires minimization of the function

\[
F = \frac{1}{2} \sigma^2 \sum_{i=1}^{n} (y_i - y_{i-1})^2 + \frac{1}{2} f(y_n)
\]

where the discretization is done using a constant time step \( \Delta t = t_f/n \) as before. Note that the control approach to this problem inserts a quadratic term through which the space is (randomly) explored.

These ideas are applied to minimize the Himmelblau function

\[
f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2
\]

which is a popular test of the performance of optimization algorithms because of its many local minima [27]. The parameters are \( R = 0.01, \sigma = 0.01, \Delta t = 1, \ t_f = 20, \) and the minimization is initialized at \((-1, -4)\).

In this example, the stochastic approach is successful and finds a much lower minimum than Newton’s method. However, since each run of the stochastic approach is random, one may find another local minimum in another run. This could be useful when one needs to explore valleys or if one suspects the existence of other local minima.

To test the reliability of the stochastic control approach, 70 experiments were performed, each starting at the same initial condition. The iterations of five such experiments are shown in Table 4. All 70 runs ended up close to the local minimum around \( x_1 = 2.5, x_2 = -2.5 \). The average value of \( F \) is an approximation to \( C \) in Eq. (18) and, with 70 experiments was found to be \( C \approx 1.75 \), which corresponds to a much lower value of \( f \) than the value \( f_{\text{Newton}} \) = 178.34, that one obtains with Newton’s method.

Table 4  Newton’s method and five runs of controlled optimization

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Newton</th>
<th>Controlled optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>260.00</td>
<td>260.00 260.00 260.00 260.00 260.00</td>
</tr>
<tr>
<td>1</td>
<td>195.36</td>
<td>268.57 266.47 241.73 337.77 257.55</td>
</tr>
<tr>
<td>2</td>
<td>180.19</td>
<td>258.45 242.92 239.19 347.24 263.20</td>
</tr>
<tr>
<td>3</td>
<td>178.43</td>
<td>259.75 268.95 258.97 360.16 243.92</td>
</tr>
<tr>
<td>4</td>
<td>178.34</td>
<td>257.44 279.41 249.28 373.97 218.95</td>
</tr>
<tr>
<td>5</td>
<td>178.34</td>
<td>214.25 243.23 224.21 347.24 201.69</td>
</tr>
<tr>
<td>6</td>
<td>178.34</td>
<td>184.70 320.29 256.08 476.23 179.34</td>
</tr>
<tr>
<td>7</td>
<td>178.34</td>
<td>161.31 331.32 230.16 394.82 157.00</td>
</tr>
<tr>
<td>8</td>
<td>178.34</td>
<td>147.39 330.16 218.89 316.97 150.73</td>
</tr>
<tr>
<td>9</td>
<td>178.34</td>
<td>144.04 255.65 175.61 245.24 150.33</td>
</tr>
<tr>
<td>10</td>
<td>101.05</td>
<td>221.05 180.92 176.98 144.97</td>
</tr>
<tr>
<td>11</td>
<td>101.05</td>
<td>221.05 174.13 180.53 105.48</td>
</tr>
<tr>
<td>12</td>
<td>48.86</td>
<td>151.54 188.66 131.86 91.42</td>
</tr>
<tr>
<td>13</td>
<td>33.05</td>
<td>115.34 171.75 101.82 67.00</td>
</tr>
<tr>
<td>14</td>
<td>29.90</td>
<td>87.37 147.43 56.51 41.29</td>
</tr>
<tr>
<td>15</td>
<td>41.34</td>
<td>62.89 104.69 35.11 10.65</td>
</tr>
<tr>
<td>16</td>
<td>31.03</td>
<td>43.69 72.71 31.48 6.08</td>
</tr>
<tr>
<td>17</td>
<td>18.12</td>
<td>22.62 42.44 14.53 1.79</td>
</tr>
<tr>
<td>18</td>
<td>1.83</td>
<td>9.195 30.55 9.52 2.61</td>
</tr>
<tr>
<td>19</td>
<td>0.46</td>
<td>1.90 0.54 0.97 1.24</td>
</tr>
</tbody>
</table>
4 Application to MCL and SLAM

Consider a mobile robot that navigates autonomously and, as it moves, collects noisy measurements about its motion as well as scans of its environment. If the scans reveal locations of features that are known to the robot, i.e., if the robot has a map of its environment, then it can localize itself on this map. This problem is known as localization. When the robot has no map of its environment, then it must construct a map while localizing itself on it, leading to the problem of SLAM [28–30]. Efficient solutions to the localization and SLAM problems are a fundamental requirement for autonomous robots, which must move through unknown environments where no global positioning data (GPS) are available, for example indoors, in abandoned mines, underwater, or on far-away planets [16,31,32]. Application of implicit sampling to localization and SLAM is the subject of Secs. 4.1 and 4.2, where the algorithms will also be tested using experimental data obtained by Nebot and colleagues at the University of Sydney [17].

4.1 MCL. The localization problem can be formulated as follows. Information about the initial state of the robot is available in the form of a PDF \( p(x_0) \), where \( x_0 \) is an \( m \)-dimensional vector whose elements are the state variables (e.g., position and velocity of the robot). A probabilistic motion model defines the PDF

\[
p(x_n|x_{n-1}, u_n) \tag{19}
\]

where \( n = 1, 2, \ldots \) is discrete time and where \( u_n \) is a \( p \)-dimensional vector of known “controls,” e.g., the odometry. A data equation describes the robot’s sensors and defines the PDF

\[
p(z_n|x_n, u_n, \Theta) \tag{20}
\]

where \( z_n \) is a \( k \)-dimensional vector whose elements are the data \((k \leq m)\) and where \( \Theta \) is the map. For example, one can use the “landmark model,” for which the map describes the coordinates of a collection of obstacles, called “landmarks;” the data are measurements of range and bearing of the position of the robot relative to a landmark. The landmark model and range and bearing data will be used in the applications below, but the algorithms described are more generally applicable.

The motion and sensor model jointly define the conditional PDF \( p(x_n|x_{n-1}, u_n, \Theta) \), where the short-hand notation \( x_{0:n} = \{x_0, x_1, \ldots, x_n\} \) is used to denote a sequence of vectors. By using Bayes’ rule repeatedly, one can derive the recursive formula

\[
p(x_n|x_{0:n-1}, u_{0:n}, \Theta) = p(x_n|x_{n-1}, u_{n-1}, \Theta) \times p(x_{n-1}|x_{0:n-1}, u_{0:n-1}, \Theta) \tag{21}
\]

where \( \frac{p(x_{n-1}|x_{0:n-1}, u_{0:n-1}, \Theta)}{p(z_n|x_{n-1}, u_n, \Theta)} \) is the denominator of Bayes’ rule.

Approximations of this PDF are used to localize the robot. For example, methods based on the Kalman filter [33] (KF) construct a Gaussian approximation which is often problematic because of nonlinearities in the model and data. Moreover, KF-based localization algorithms can diverge, for example, in multimodal scenarios (i.e., if the data are ambiguous) or during relocalization (i.e., if the data are ambiguous) or during relocalization or during relocalization. Localizations algorithms can diverge, for example, in multimodal scenarios (i.e., if the data are ambiguous) or during relocalization (i.e., if the data are ambiguous) or during relocalization. Efficient solutions to the localization and SLAM problems are a fundamental requirement for autonomous robots, which must move through unknown environments where no global positioning data (GPS) are available, for example indoors, in abandoned mines, underwater, or on far-away planets [16,31,32]. Application of implicit sampling to localization and SLAM is the subject of Secs. 4.1 and 4.2, where the algorithms will also be tested using experimental data obtained by Nebot and colleagues at the University of Sydney [17].

4.1.1 Implementation for the University Car Park Data Set. The University Car Park data set [17] is used to demonstrate the efficiency of MCL with implicit sampling. The scenario is as follows. A robot is moving around a parking lot and steering and speed data are collected via a wheel encoder on the rear left wheel and a velocity encoder. An outdoor laser (SICK LMS 221) is mounted on the front bull-bar and is directed forward and horizontal (see Ref. [36] for more information on the hardware). The laser returns measurements of relative range and bearing to different features. Speed and steering data are the controls of a kinematic model of the robot and the laser data are used to update this model’s predictions about the state of the robot.

The motion model is the forward Euler discretization of the continuous time two-dimensional-model in Ref. [36]

\[
x_{n+1} = x_n + R(x_n, u_n)\delta + \Delta B_n, \quad \Delta B_n = N(0, \Sigma_1) \tag{22}
\]

where \( R(x_n, u_n) \) is a three-dimensional vector function, \( \delta \) is the time step, and \( \Sigma_1 \) is a given covariance matrix. The data equation is

\[
z_n = h(x_n, \Theta) + V_n, \quad V_n \sim N(0, \Sigma_2) \tag{23}
\]

where \( h \) is a two-dimensional vector function that maps the position of the robot to relative range and bearing and where \( \Sigma_2 \) is a \( 2 \times 2 \) diagonal matrix (see Ref. [36] and the Appendix for the various model parameters).

With the above equations for motion model and data, the function \( F^j \) of implicit sampling in Eq. (21) become

\[
F^j(x_n) = (x_n - f^j_n)^T (\Sigma_1)_n^{-1} (x_n - f^j_n) + \sum_i \left(h(x_n) - z^j_{im}\right)^T \Sigma_2^{-1} (h(x_n) - z^j_{im}) \tag{24}
\]

where \( f^j_n = X^j_{n-1} + R(X^j_{n-1}, u_n)\delta \) and where \( z^j_{im} \) denotes the measurement for the \( i \)th landmark at time \( n \). These \( F^j \)'s are minimized with Newton’s method and samples are generated by solving a quadratic equation as explained in Sec. 2. The gradient and Hessian in Newton’s method were computed analytically (see the
Appendix). If uneven weights were observed, a “resampling” was done using Algorithm 2 in Ref. [37]. Resampling replaces samples with a small weight with samples with a larger weight without introducing significant bias. If no laser data are available, all samples evolve according to the model equation (19).

The results of MCL with implicit sampling are shown in the left panel of Fig. 6. The “ground truth” shown here is the result of an MCL run with GPS data (these GPS data however are not used in implicit sampling); the large dots are the positions of the landmarks. The reconstruction by the MCL algorithm (dashed line) is an approximation of the conditional mean, which is obtained by averaging the samples. One observes that MCL with implicit sampling gives accurate estimates whenever laser data are available. After periods during which no laser data were available, one can observe a strong “pull” toward the true state trajectory, as soon as data become available, as is indicated by the jumps in the trajectory, e.g., around \( x = 0, y = 0 \).

The efficiency of MCL with implicit sampling is studied by running the algorithm with 10, 20, 40, 80, 100, and 150 samples, i.e., with increasing computational cost, and comparing the reconstruction of the MCL with the true path. The error is measured by the Euclidean norm of the difference between the “ground truth” (see above) and the conditional PDF of interest is thus

\[
p(x_0, \Theta | z_{0:n}, u_{0:n}, g_0) \quad (23)
\]

where the map, \( \Theta \), is a variable (not a given parameter as in MCL); the variable \( g_{0:n} \) is the “data association,” i.e., it maps the data to the known features, or creates a new feature if the data are incompatible with current features [38,39]. Here it is assumed that the data association is done by another algorithm, so that \( g_{0:n} \) in Eq. (23) can be assumed to be given (in the numerical implementation in Sec. 4.2.1, a maximum-likelihood algorithm is used [15]).

Note that the SLAM state-vector is different from the state of the localization problem: it is the number of variables needed to describe the robot’s position, \( x \), and the coordinates of the features of the map, \( \Theta \). If the map contains many features (which is typically the case), then the state dimension is large and this makes KF based SLAM prohibitively expensive. The reason is that KF SLAM requires dense matrix operations on matrices of size \( N \) (where \( N \) is the number of features), due to correlations between the robot’s position and the features [38]. KF SLAM thus has \( N^2 \) memory requirements which make it infeasible for realistic \( N \).

MC approaches to SLAM reduce the computational cost by exploring conditional independencies [40]. In particular, the

<table>
<thead>
<tr>
<th>Table 5 Simulation results for MCL</th>
</tr>
</thead>
<tbody>
<tr>
<td># of samples</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td># of samples</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>150</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>250</td>
</tr>
<tr>
<td>300</td>
</tr>
</tbody>
</table>

It is clear that the both MCL algorithms converge to the same error, since both algorithms converge to the conditional mean as the number of samples (and hence, the computational cost) goes to infinity. However the convergence rate of MCL with implicit sampling is faster, which can be seen from the right panel of Fig. 6, where the computing time is plotted as a function of the error. In this figure, the area between the lines corresponds to the improvement of implicit sampling over the standard method, and it is clear that implicit sampling is more efficient than the standard method. The improvement is particularly pronounced if a high accuracy is required, in which case MCL with implicit sampling can be orders of magnitudes faster than the standard method.
various features of the map conditioned on the robot path are independent, which implies the factorization

$$p(\Theta|x_{0:N}, z_{0:N}, u_{0:N}, \eta_{0:N}) = \prod_{k=1}^{N} p(\hat{p}^{k}|x_{0:N}, z_{0:N}, u_{0:N}, \eta_{0:N})$$

where $\hat{p}^{k}$, $k = 1, \ldots, N$ are the features of the map [15,41]. This factorization makes it possible to update one feature at a time and thus reduces memory requirements. Moreover, “online” SLAM will be considered here, i.e., the map and the robot’s position are constructed recursively as data become available, using

$$p(x_{0:N}, \Theta|z_{0:N}, u_{0:N}, \eta_{0:N}) \propto p(x_{0:N-1}, \Theta|z_{0:N-1}, u_{0:N-1}, \eta_{0:N-1}) \times p(x_{n}|x_{n-1}, u_{n})p(z_{n}|\theta_{n}, x_{n}, \eta_{n})$$

(24)

where $\theta_{n}$ is the feature observed at time $n$. Alternatively, one can wait and collect all the data, and then assimilate this large data set in one sweep; such a “smoothing” approach is related to graph based SLAM (see, e.g., Ref. [42]) but will not be discussed further in this paper.

For online probabilistic SLAM, one assumes that $M$ weighted samples $\{x_{0:n-1}^{(i)}, \Theta^{(i)}, W_{n-1}^{(i)}\}$ approximate

$$p(x_{0:n-1}, \Theta|z_{0:n-1}, u_{0:n-1}, \eta_{0:n-1})$$

at time $n-1$ and then updates the samples to time $n$ by applying importance sampling to

$$p(x_{n}|x_{n-1}, u_{n})p(z_{n}|\theta_{n}, x_{n}, \eta_{n})$$

Here, it is assumed that only one feature is observed at a time (which is realistic [15]); however, the extension to observing multiple features simultaneously is straightforward.

Various importance functions have been considered and tested in the literature. For example, the fastSLAM algorithm [43] was shown to be problematic in many cases due to the poor distribution of its samples [15,44]. The reason is that the samples are generated by the motion model and, thus, are not informed by the data. As a result, the samples are often unlikely with respect to the data. The fastSLAM 2.0 algorithm [15,44] ameliorates this problem by making use of an importance function that depends on the data. The algorithm was shown to outperform fastSLAM and was proven to converge (as the number of samples goes to infinity) to the true SLAM posterior for linear models. The convergence for nonlinear models is not well understood (because of the use of extended KFs (EKFs) to track and construct the map). Here, a new SLAM algorithm with implicit sampling is described. The algorithm converges for nonlinear models at a computational cost that is comparable to the cost of fastSLAM 2.0.

Online SLAM with implicit sampling requires that implicit sampling is applied to the $M$ functions $F^i$ (one per sample), whose variables are the position at time $t_n$, $x_n$, and the location of the feature $\theta_n$ observed at time $t_n$

$$F^i(x_n, \theta_n) = -\log p(x_n|x_{n-1}^{(i)}, u_n)p(z_n|\theta_n^{(i)}, x_n, \eta_n^{(i)})$$

the position of the $j$th sample at time $t_{n-1}$ is a parameter. As in MCL, one needs $M$ functions here due to the recursive problem formulation. The minimization problems of implicit sampling are

$$\hat{\theta}^i_n = \min_{\theta_n} F^i(x_n, \theta_n)$$

and samples $\{x_n^{(i)}, \hat{\theta}^i_n\}$ are generated by solving the stochastic equation (1). One can use the same technique for solving these equations as described in Sec. 2. Moreover, if the feature $\theta_n$ is already known, the SLAM algorithm reduces to the MCL algorithm with implicit sampling described above.

Note that the memory requirements of SLAM with implicit sampling are linear in the number of features, because the algorithm makes use of the conditional independence of the features given the robot path, so that, at each step, only one feature needs to be considered. Moreover, SLAM with implicit sampling converges to the true SLAM posterior (under the usual assumptions about the supports of the importance function and the target PDF [45]) as the number of samples goes to infinity; this convergence is independent of linearity assumptions about the model and data equations. Convergence for fastSLAM and fastSLAM 2.0, however, can only be proven for linear Gaussian models [15].

4.2.1 Numerical Experiments: The applicability and efficiency of SLAM with implicit sampling is demonstrated with numerical experiments that mimic the University Car Park data set [17]. Here speed and steering data of Ref. [17] are used, however the laser data are replaced by synthetic data, because the data in Ref. [17] are too sparse (in time) for successful online SLAM (even EKF SLAM, often viewed as a benchmark, does not give satisfactory results).

In these synthetic data experiments, a virtual laser scanner returns noisy range and bearing measurements of features in a half-circle with 15 m radius. If the laser detects more than one feature, one is selected at random and returned to the SLAM algorithm. Since each synthetic data set is a random event, 50 synthetic data sets are used to compute the average errors for various SLAM algorithms. These errors are defined as the norm of the difference of the ground truth and the conditional mean computed with a SLAM algorithm. The performances of EKF SLAM, fastSLAM, fastSLAM 2.0, and SLAM with implicit sampling are compared using these synthetic data sets. All SLAM methods make use of the same maximum likelihood algorithm for the data association. A Newton method was implemented for the optimization in implicit sampling, and the quadratic approximation of $F$ was used to solve the algebraic equation (1). A typical outcome of a numerical experiment is shown in the left panel of Fig. 7, where the true path and true positions of the landmarks as well as their reconstructions via SLAM with implicit sampling (100 samples) are plotted.

The results of 50 numerical experiments with fastSLAM, fastSLAM 2.0, and SLAM with implicit sampling are shown in Table 6. It was observed in this example that EKF SLAM gives an average error of 3.89% at an average computation time of 0.43 ms and, thus, is computationally efficient and accurate. However, computational efficiency disappears in scenarios with larger maps due to the $O(N^2)$ memory requirements ($N$ is the number of features in the map). The error of fastSLAM 2.0 (whose memory requirements scale linearly with $N$) decreases with the number of samples, and it seems as if the fastSLAM 2.0 solution converges (as the number of samples and, thus, the computation time increases) to the EKF solution. This is intuitive because fastSLAM 2.0 uses EKFs to track the map. The fastSLAM algorithm, on the other hand, converges to an error that is larger than in EKF SLAM or fastSLAM 2.0. SLAM with implicit sampling converges to an error that is smaller than in EKF SLAM. The reason is as follows: implicit sampling requires no linearity assumptions and, therefore, the empirical estimate it produces converges (as the number of samples goes to infinity) to the true SLAM posterior.

Moreover, the convergence rate of SLAM with implicit sampling is faster than for any other SLAM method, due to the data informed importance function. Thus, the approximation of the conditional mean (the minimum mean square error estimator) one obtains with implicit sampling is accurate even if the number of samples is relatively small. As a result, SLAM with implicit sampling is more efficient than the other SLAM algorithms considered here. This is illustrated in the right panel of Fig. 7, where the computing time is shown as a function of the average error. As in MCL (see above), the area between the various curves indicates the improvement in efficiency. Here implicit sampling is the most efficient method, giving a high accuracy (small error) at a small computational cost. Moreover, SLAM with implicit sampling can
achieve an accuracy which is higher than the accuracy of all other methods (including EKF SLAM), because it is a convergent algorithm.

5 Conclusions

Implicit sampling is a MC scheme that localizes the high-probability region of the sample space via numerical optimization, and then produces samples in this region by solving algebraic equations with a stochastic right-hand side. The computational cost per sample in implicit sampling is larger than in many other MC sampling schemes that randomly explore the sample space. However, the minimization directs the computational power toward the relevant region of the sample space so that implicit sampling often requires fewer samples than other MC methods. There is a tradeoff between the cost-per-sample and the number of samples and this tradeoff was studied in this paper in the context of three applications: MCL and probabilistic online SLAM in robotics, as well as path integral control.

In path integral control one solves the stochastic HJB equation with a MC solver. This has the advantage that no grid on the state space is needed and the method is therefore (in principle) applicable to control problems of relatively large dimension. Implicit sampling was applied in this context to speed up the MC calculations. The applicability of this approach was demonstrated on two test problems. Path integral control was also used to find local minima in nonconvex optimization problems.

An implementation of implicit sampling for MCL was tested and it was found that implicit sampling performs better than standard MCL (in terms of computing time and accuracy), especially if the data are accurate and the motion model is noisy (which is the case typically encountered in practice [16]). Similarly, implicit sampling for SLAM outperformed standard algorithms (fastSLAM and fastSLAM 2.0). Under mild assumptions, but for nonlinear models, SLAM with implicit sampling converges to the true SLAM posterior as the number of samples goes to infinity, at a computational cost that is linear in the number of features of the map.

Acknowledgment

I thank Professor Alexandre J. Chorin of UC Berkeley for many interesting technical discussions and for bringing path integral control to my attention. I thank Dr. Robert Saye of Lawrence Berkeley National Laboratory for help with proofreading this manuscript. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Contract No. DE-AC02005CH11231, and by the National Science Foundation under Grant No. DMS-1217065.

Appendix

A kinematic model of the robot is described in Ref. [46] and shown for convenience in Fig. 8. The robot is controlled by the speed \( v_c \) and the steering angle \( \alpha \). It can be shown that the derivatives of the position and orientation of the axle of the robot are

<table>
<thead>
<tr>
<th>Samples</th>
<th>fastSLAM</th>
<th>fastSLAM 2.0</th>
<th>Implicit sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU time in ms</td>
<td>Error in %</td>
<td>CPU time in ms</td>
</tr>
<tr>
<td>10</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>20</td>
<td>5.55</td>
<td>17.7</td>
<td>8.36</td>
</tr>
<tr>
<td>50</td>
<td>13.6</td>
<td>14.8</td>
<td>20.5</td>
</tr>
<tr>
<td>100</td>
<td>26.9</td>
<td>12.0</td>
<td>40.7</td>
</tr>
<tr>
<td>200</td>
<td>52.7</td>
<td>10.8</td>
<td>82.1</td>
</tr>
<tr>
<td>400</td>
<td>105.2</td>
<td>7.64</td>
<td>164.0</td>
</tr>
<tr>
<td>500</td>
<td>133.2</td>
<td>8.67</td>
<td>346.7</td>
</tr>
<tr>
<td>800</td>
<td>213.2</td>
<td>10.4</td>
<td>—</td>
</tr>
<tr>
<td>1000</td>
<td>280.1</td>
<td>8.78</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 6 Simulation results for SLAM

Fig. 7 Simulation results for SLAM. Left: ground truth and reconstruction via SLAM with implicit sampling (100 samples). Right: computing time as a function of the average error of fastSLAM, fastSLAM 2.0, and SLAM with implicit sampling.
so that the forward model (without noise) is

\[
\frac{dx}{dt} = \nu_c \cos(\beta), \quad \frac{dy}{dt} = \nu_c \sin(\beta), \quad \frac{db}{dt} = \frac{\nu_c}{L} \tan(z)
\]

The velocity \( \nu_c \), however, is measured at the rear left wheel and, thus, must be translated to the axle

\[
\nu_c = \frac{L\nu_l}{L - \tan zH}
\]

where \( L \) is the wheelbase of the vehicle, \( H \) is the horizontal offset of the laser, \( z \) is the angle of the laser with respect to the horizontal, and \( \nu_l \) is the velocity of the left wheel.

The position of the laser can be obtained from the position of the axle by using

\[
x = x_c + a \cos(\beta) - b \sin(\beta), \quad y = y_c + a \sin(\beta) + b \cos(\beta)
\]

so that the forward model (without noise) is

\[
\begin{align*}
\frac{dx}{dt} &= \nu_c \cos(\beta) - \tan(z(\alpha \sin(\beta) - b \cos(\beta))) \\
\frac{dy}{dt} &= \nu_c \sin(\beta) + \tan(z(\alpha \cos(\beta) - b \sin(\beta))) \\
\frac{db}{dt} &= \frac{\nu_c}{L} \tan(z)
\end{align*}
\]

which, in a more compact notation, can be written as

\[
\frac{dx}{dt} = R(x, u)
\]

where \( x = (x, y, \beta)^T \) and \( u = (\nu_c, z)^T \). The uncertainty in the model is due to errors in the inputs and in the model [46]

\[
\begin{align*}
\frac{dx}{dt} &= R(x, u(t))dt + QdW \quad \frac{dW}{dt} = \nu_c(\tan(z) - \tan(z(t))) \\
\hat{u}(t)dt &= u(t)dt + \nu_c(\tan(z) - \tan(z(t)))
\end{align*}
\]

where \( W \) and \( W(t) \) are two- and three-dimensional vectors whose components are independent Brownian motions, \( P \) and \( Q \) are real \( 2 \times 2 \) and \( 3 \times 3 \) matrices and \( \nu_c(t) \) is the unperturbed control signal. The above equations are linearized in \( \hat{u} \) to obtain

\[
dx = R(x, u)dt + \frac{dR}{du}PdW_u + QdW
\]

and the stochastic forward Euler method [21] with time step \( \delta \) is used to discretize the equations

\[
x^{n+1} = x^n + R(x^n, u^n)\delta + \Delta W^n, \quad \Delta W^n = \mathcal{N}(0, \Sigma^n)
\]

where

\[
\Sigma^n = \delta \left( \frac{dR}{du} p p^T \frac{dR}{du} + QQ^T \right)
\]

and \( x^n = x(n\delta), u^n = u(n\delta) \). Table 7 lists the numerical values of all parameters of the probabilistic motion model. The steering data \( x \) and velocity data \( v_c \) are taken from the data set [17].

The data are measurements of range and bearing of the features in the map \( \theta \) relative to the robot and are obtained by the laser. Only “high intensity points” from the data set [17] are used. The laser is modeled by the data equation

\[
z^n = h(x^n) + V^n, \quad V^n \sim \mathcal{N}(0, \Sigma_z)
\]

where \( \Sigma_z \) is a \( 2 \times 2 \) diagonal matrix whose elements are \( \sigma_x = 0.05 \) and \( \sigma_y = 0.05\pi/180 \) and

\[
h(x^n) = \left( \begin{array}{c}
\frac{\|x^n - m^n|}{\sigma_x} \\
\tan^{-1} \left( \frac{m_{z1} - x_{z1}}{m_{z2} - x_{z2}} \right)
\end{array} \right)
\]

where \( z_{i} \) is the \( i \)th element of the vector \( z^n \), the “true” robot position, and \( m_{z1}, m_{z2} \) are the \( x \)- and \( y \)-coordinates of the \( i \)th element of a landmark; the norm \( \| \cdot \| \) is the Euclidean norm.

The gradient and Hessian of \( F_j \) in Eq. (22) are

\[
\nabla F_j = (\Sigma^n)^{-1}(x - f^n) + \sum_{i=1}^{n} \left( \begin{array}{c}
x_1 - m_1 \frac{(r_1 - z_1)}{\sigma^2} \\
x_2 - m_2 \frac{(r_2 - z_2)}{\sigma^2} \\
x_3 - m_3 \frac{(r_3 - z_3)}{\sigma^2}
\end{array} \right)
\]

\[
H_j = (\Sigma^n)^{-1} + \sum_{i=1}^{n} \bar{H}_i
\]

where the elements of the \( 3 \times 3 \) matrices \( H_j \) are

<table>
<thead>
<tr>
<th>Table 7</th>
<th>Model parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Parameter</td>
</tr>
<tr>
<td>Wheel base</td>
<td>( L )</td>
</tr>
<tr>
<td>Width of the car</td>
<td>( H )</td>
</tr>
<tr>
<td>Horizontal offset of laser</td>
<td>( b )</td>
</tr>
<tr>
<td>Rear axis to laser</td>
<td>( a )</td>
</tr>
<tr>
<td>Time step</td>
<td>( \delta )</td>
</tr>
<tr>
<td>Covariance of model noise</td>
<td>( Q )</td>
</tr>
<tr>
<td>Covariance of noise in controls</td>
<td>( P )</td>
</tr>
</tbody>
</table>
\[
H_{11} = \frac{r_1 - z_1}{\sigma_r^2} + \frac{(x_1 - m_1)^2}{\sigma_x^2} + \frac{(y_1 - m_y)^2}{\sigma_y^2} \\
H_{12} = \frac{x_1 - m_1}{\sigma_x^2} + \frac{y_1 - m_y}{\sigma_y^2} \\
H_{13} = \frac{1}{\sigma_x^2}
\]

and where
\[
\vartheta = \left( \frac{m_y - y_1}{m_x - x_1} \right) - \tan^{-1}\left( \frac{m_x - x_1}{m_y - y_1} \right)
\]

References


\[\text{References}\]