Nonlinear Model Predictive Control Using Sampling and Goal-Directed Optimization

Damion D. Dunlap, Charmane V. Caldwell, and Emmanuel G. Collins Jr.

Abstract— **In this paper a novel method called Sampling-Based Model Predictive Control (SBMPC) is proposed as an efficient MPC algorithm to generate control inputs and system trajectories. The algorithm combines the benefits of sampling-based motion planning with MPC while avoiding some of the major pitfalls facing both traditional sampling-based planning algorithms and traditional MPC. The method is based on sampling (i.e., discretizing) the input space at each sample period and implementing a goal-directed optimization method (e.g., $A^\star$) in place of standard numerical optimization. This formulation of MPC readily applies to systems with nonlinear dynamics and avoids the local minima which can limit the performance of MPC algorithms implemented using traditional, derivative-based, nonlinear programming. The SBMPC algorithm is compared with a more standard online MPC algorithm using cluttered environment navigation for an Ackerman steered vehicle and a set point problem for a nonlinear, continuous stirred-tank reactor (CSTR).**

I. INTRODUCTION

Motivated largely by the problem of trajectory generation for autonomous vehicles, this paper attempts to add to the growing literature on nonlinear model predictive control (NMPC) [1], [2]. The proposed methodology, called Sampling-Based Model Predictive Control (SBMPC), is a NMPC algorithm that can avoid local minimum, has strong convergence properties based on the $LPA^\star$ optimization convergence proof [15], has physically intuitive tuning parameters, and has been experimentally observed to be fast enough to implement in real-time for at least some fast systems. The concept behind SBMPC was first presented in [3]. However, this paper presents a much more developed and tested SBMPC methodology. SBMPC utilizes fast replanning to achieve robustness, which can be improved by using a closed loop model to obtain predictions that are less sensitive to model uncertainties [4], [5].

D. Dunlap is with the Department of Mechanical Engineering and the Center for Intelligent Systems, Control and Robotics (CISCOR) of the Florida A&M University - Florida State University College of Engineering (FAMU-FSU COE), 2525 Pottsdamer St., Tallahassee, FL 32310, USA dunlap@eng.fsu.edu

C. Caldwell is with the Department of Electrical and Computer Engineering and CISCOR of the FAMU-FSU COE cvcaldwe@eng.fsu.edu

E. Collins is Faculty with the Department of Mechanical Engineering and CISCOR of the FAMU-FSU COE ecollins@eng.fsu.edu

Prepared through collaborative participation in the Robotics Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD 19-01-2-0012. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. This work is also supported by the National Science Foundatation under award CMMI-0927040

Although SBMPC is generally formulated and can be utilized for any NMPC or linear MPC problem, a focus of this presentation is autonomous vehicle trajectory planning. At least two MPC approaches have been previously developed for this planning problem. One method [6] uses mixed-integer programming and a process called "cost-to-go" to reduce the prediction horizon and approximate the remainder of the path using a graph search. Each vehicle is modeled as a (linear) point mass with linear constraints on the speed and turn rate. In contrast, the approach of [7] does allow for nonlinear vehicle models and uses MPC coupled with the potential field method to allow the vehicle to maneuver around obstacles and other autonomous vehicles. The use of a potential field is used to trade performance for computational efficiency.

As its name implies, SBMPC is dependent upon the concept of sampling, which has arisen as one of the major paradigms for robotic motion planning [8]. Sampling, which is explained in Subsection II-A, is the mechanism used to trade performance for computational efficiency. SBMPC employs quasi random samples of the space. Properly designed sampling algorithms have the theoretical property that if the sampling is dense enough, the sampling algorithm will find a solution when it exists (i.e., it has some type of completeness [9]). Completeness is akin to MPC stability properties based on extending the prediction horizon to infinity [10]. Unlike MPC, which views the system behavior through the system inputs, the vast majority of previously developed sampling methods plan in the output space and try to find inputs that connect points in the output space. As is discussed in Subsection II-A, this can be highly inefficient.

In general there are various types of MPC that have been formulated throughout the years. For the purpose of this paper, the term traditional MPC only considers model predictive control methods that use numerical optimization online to determine the control inputs. SBMPC is an algorithm that, like traditional MPC, is based on viewing the system through its inputs. However, unlike previous MPC methods, it uses sampling to provide the trade-off between performance and computational efficiency. Also, in contrast to previous MPC methods, it does not rely on linear or nonlinear programming, or heuristic global optimization approaches such as evolutionary algorithms. Instead it uses a goal-directed optimization algorithm derived from $LPA^\star$ [15], an incremental $A^\star$ algorithm [8]. In general, goal-directed algorithms (see Subsection II-B) are efficient for problems with the type of discretization that arises from

sampling and guarantee a solution that is globally optimal, subject to the employed sampling.

As mentioned previously, early ideas for SBMPC first appeared in [3]. Parallel, but independent work has resulted in a closely related motion planning algorithm [4] that has been implemented in real time on a vehicle that competed in and successfully completed the DARPA Urban Challenge (DUC). Like SBMPC, the algorithm of [4] uses sampling of the system inputs and optimization to develop the Closed-Loop Rapidly-exploring Random Tree (CL-RRT) algorithm. This algorithm was developed to take advantage of a pure-pursuit controller, which enables the inputs to the controller to contain the desired position of the vehicle. In practice, having the desired vehicle position as part of the controller input obviates the position clustering phenomenon that other researchers have encountered when attempting to use input sampling. SBMPC can be applied to closed-loop models that use pure-pursuit controllers. However, SBMPC has an implicit state gridding mechanism that enables it to avoid clustering, even when the inputs do not contain the desired vehicle positions (or more generally the desired system outputs). Although both [4] and this research were motivated by the problem of autonomous vehicle guidance, the focus of [4] was the development of specific algorithm features to enable a vehicle to meet the specifications of the DUC. In contrast, the research here seeks to make explicit connections with NMPC in order to provide a bridge that can be used to extend the applicability of the sampling methods to problem domains that do not necessarily involve robotic motion planning.

This paper is arranged as follows. Section II provides the necessary background for SBMPC and serves as a tutorial on sampling methods and goal-directed optimization. Section III presents the SBMPC algorithm and discusses implementation issues. Section IV compares SBMPC with standard NMPC algorithms using cluttered environment navigation for an Ackerman steered vehicle and a set point problem for a nonlinear continuous stirred-tank reactor. Finally, Section V presents conclusions.

II. Fundamental Concepts of Sampling-Based Planning

Sampling-based motion planning algorithms include RRTs [11], probability roadmaps [12], and randomized $A^\star$ algorithms [13]. A common feature of each of those algorithms to date is that they work in the configuration space of the robot[1] and use various strategies for generating samples (i.e., random or pseudo-random points). In essence, as shown in Fig. 1, sampling-based motion planning methods work by using sampling to construct a tree that connects the root with a goal region. The general purpose of sampling is to cover the space such that the samples are uniformly distributed, while minimizing gaps and clusters [9]. This can prove to be challenging if an appropriate sampling technique is not used. The sampling method used in this research is based on Halton points [14] because they can be added to existing Halton points while maintaining good uniformity and are thus more suitable for generating algorithms that have the flexibility to add samples when needed.

Most online sampling-based planning algorithms follow this general framework:

1) **Initialize:** Let $G(V;E)$ represent a search graph where V contains at least one vertex (i.e., node), typically the start vertex and E does not contain any edges.
2) **Vertex Selection Method (VSM):** Select a vertex u in V for expansion.
3) **Local Planning Method (LPM):** For some $u_{new} \in C_{free}$ (free states in the configuration space) and attempt to generate a path $\tau_s : [0,1] \rightarrow: \tau(0) = u$ and $\tau(1) = u_{new}$. The path must be checked to ensure that no constraints are violated. If the LPM fails, then go back to Step 2.
4) **Insert an Edge in the Graph:** Insert τ_s into E, as an edge from u to u_{new}. Insert u_{new} into V if it doesnt already exist.
5) **Check for a Solution:** Check G for a solution path.
6) **Return to Step 2:** Repeat unless a solution has been found or a failure condition has been met.

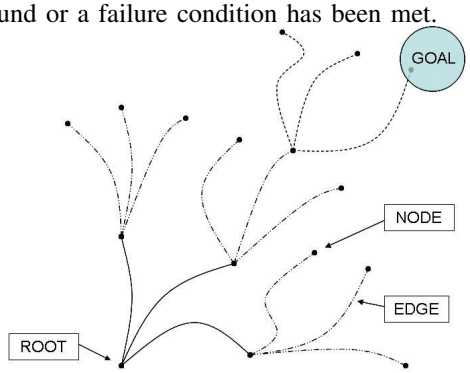

Fig. 1. A tree that connects the root with a goal region.

A. Comparison of Input and Output Sampling

There are two primary disadvantages to using output (i.e., configuration space) sampling as is commonly done in robotics. The first limitation lies within the VSM, where the algorithm must determine the most ideal vertex to expand. This selection is typically made based on the proximity of vertices in the graph to a sampled output point, and involves a potentially costly nearest neighbor search. The LPM presents the second, and perhaps more troublesome problem, which is determining an input that connects a newly sampled node to the current node. This problem is essentially a two-point boundary value problem (BVP) that connects one output or state to another. There is no guarantee that such an input exists. Also, for systems with complex dynamics, the search itself can be computationally expensive, which leads to a computationally inefficient planner. In contrast, when the input space is sampled as proposed in this paper, the need for a nearest-neighbor search is eliminated and the LPM is

[1] The configuration space corresponds to the degrees of freedom that are being considered in the motion planning problem. For example, in planar path planning for an autonomous ground vehicles the configuration space is typically the horizontal x-y coordinates of the vehicle center of mass. The configuration space is essentially some subset of the state space of a particular model of the robot being considered.

reduced to the integration of a system model and therefore only generates outputs that are achievable by the system.

B. Goal Directed Optimization

There is a class of discrete optimization techniques that have their origin in graph theory and have been further developed in the path planning literature. In this paper these techniques will be called *goal-directed optimization* and refer to graph search algorithms such as Dijkstra's algorithm and the $A^\star$, $D^\star$, and $LPA^\star$ algorithms [8], [15]. Given a graph, these algorithms find a path that optimizes some cost of moving from a start node to some given goal. In contrast to discrete optimization algorithms such as branch-and-bound optimization [16], which "relaxes" continuous optimization problems, the goal-directed optimization methods are inherently discrete, and have often been used for real-time path planning. In the next subsection, goal-directed optimization is further contrasted with standard MPC optimization.

C. Comparison of MPC Optimization and Goal-Directed Optimization

Although not commonly recognized, goal-directed optimization approaches are capable of solving control problems for which the ultimate objective is to generate an optimal trajectory and control inputs to reach a goal (or set point) while optimizing a cost function; hence, they apply to terminal constraint optimization problems and set point control problems. To see this, consider the tree graph of Fig. 1. Each node of this tree can correspond to a system state and the entire tree may be generated by integrating sampled inputs to a system model. Assume that the cost of a trajectory is given by the sum of the cost of the corresponding edges (i.e., branches), where the cost of each edge is dependent not only on the states it connects but also the inputs that are used to connect those states. The use of the system model can be viewed simply as a means to generate the directed graph and associated edge costs, therefore maintaining the proofs of completeness and optimality associated with that particular goal-directed optimization algorithm. This fundamental insight is the basis of SBMPC.

To understand the relationship between sampling-based algorithms and MPC optimization, it is essential to pose sampling-based path planning as an optimization problem. To illustrate this point, we claim that *, subject to the constraints of the sampling,* a goal-directed optimization algorithm can effectively solve the mixed integer nonlinear optimization problem,

$$\min_{\{u(k),\ldots,u(k+N-1)\},N} J = \sum_{i=0}^{N} \parallel y(k+i+1) - y(k+i) \parallel_{Q(i)} + \sum_{i=0}^{N-1} \parallel \Delta u(k+i) \parallel_{S(i)} \quad (1)$$

subject to the system equations,

$$x(k+i) = f(x(k+i-1), u(k+i-1)), \quad (2)$$

$$y(k) = g(x(k)), \quad (3)$$

and the constraints,

$$\parallel y(k+N) - \mathbf{G} \parallel \leq \epsilon, \quad (4)$$

$$x(k+i) \in \mathbf{X}_{free} \ \forall \ \ i \leq N, \quad (5)$$

$$u(k+i) \in \mathbf{U}_{free} \ \forall \ \ i \leq N, \quad (6)$$

where $\Delta u(k+i) = u(k+i) - u(k+i-1)$, $Q(i) \geq 0$, $S(i) \geq 0$, and $\mathbf{X}_{free}$ and $\mathbf{U}_{free}$ represent the states and inputs respectively that do not violate any of the problem constraints. The term $\parallel y(k+i+1) - y(k+i) \parallel_{Q(i)} + \parallel \Delta u(k+i) \parallel_{S(i)}$ represents the edge cost of the path between the current predicted output $y(k+i)$ and the next predicted output $y(k+i+1)$. The goal state G is represented as a terminal constraint as opposed to being explicitly incorporated into the cost function. Goal-directed optimization methods implicitly consider the goal through the use of a function that computes a rigorous lower bound of the cost from a particular state to G. This function, often referred to as an "optimistic heuristic" in the robotics literature, is eventually replaced by actual cost values based on the predictions and therefore does not appear in the final cost function. The cost function can be modified to minimize any metric as long as it can be computed as the sum of edge costs.

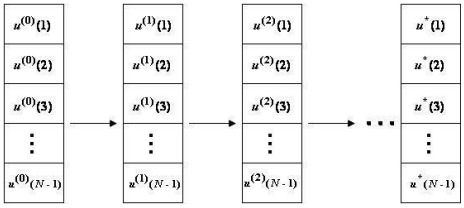

Fig. 2. Standard MPC optimization

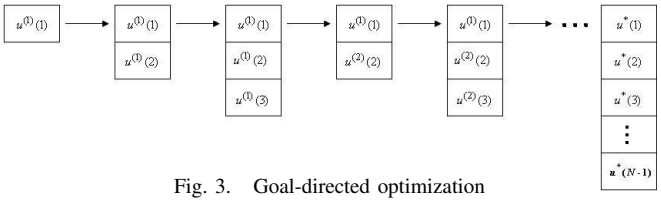

Fig. 3. Goal-directed optimization

Goal-directed optimization approaches operate differently than traditional optimization methods utilized in MPC as illustrated by contrasting Figs. 2 and 3, which contain the term $u^{(i)}(k)$, where u denotes the control input vector (the optimization parameter), i denotes the i^{th} iteration, and k denotes the time step. Referring to Fig. 2, standard MPC optimization begins with an initial sequence $\{u^{(0)}(k)\}$ of control inputs (denoted by the left array) and iterates (denoted by the arrows) until it determines the optimal control trajectory $\{u^*(k)\}$ (denoted by the right-most array). Referring to Fig. 3, goal-directed optimization, in contrast to traditional MPC optimization, sequentially computes the next control inputs and backs up when needed as illustrated by the iterations corresponding to the 3rd, 4th and 5th arrays. This feature enables it to avoid local minima. It converges to the sequence $\{u^*(k)\}$ (denoted by the right-most array), which is the optimal solution subject to the sampling, whereas a nonlinear programming method may get stuck at a local minimum.

III. SAMPLING-BASED MODEL PREDICTIVE CONROL

SBMPC is a method that can address NMPC problems. It effectively reduces the problem size of MPC by sampling the inputs of the system. The method also replaces the traditional MPC optimization phase with $LPA^\star$, an algorithm derived from $A^\star$ that can replan quickly (i.e., it is incremental). The objective is to determine a sequence of control inputs that cause the system to achieve a given set point while solving the optimization problem (1) - (6). SBMPC optimization differs from the typical application of $A^\star$ in that the nodes are configurations in the output space and an edge is the control input that links two output nodes. SBMPC retains the computational efficiency and has the convergence properties of $LPA^\star$ [15] under the conditions discussed in Subsection III-E, while avoiding some of the computational bottlenecks associated with sampling-based motion planners. In particular, the LPM (see Subsection II-A) of sampling-based motion planners is simplified from what is effectively a two-point BVP by sampling in the input space and integrating to determine the next output node. Although most sampling-based planners use proximity to a random or heuristically biased point as their VSM, the proposed method bases the vertex selection on an $A^\star$ criterion, which involves computing the cost to the node and an optimistic (i.e., lower bound) prediction of the cost from the node to the goal; this eliminates the need for a potentially costly nearest neighbor search while promoting advancement towards the goal.

A. The Implicit State Grid

Although extension of several of the existing sampling-based paradigms can lead to input sampling algorithms like SBMPC [8], input sampling has not been used in most planning research. This is most likely due to the fact that input sampling is seen as being inefficient because it can result in highly dense samples in the configuration space since input sampling does not inherently lead to a uniformly discretized state space, such as a uniform grid. This problem is especially evident when encountering a local minimum problem associated with the $A^\star$ algorithm, which can occur when planning in the presence of a large obstacle while the goal is on the other side of the obstacle. This situation is considered in depth for discretized 2D path planning in the work of [13], which discusses that the $A^\star$ algorithm must explore all the states in the neighborhood of the local minimum, shown as the shaded region of Fig. 4, before progressing to the final solution. The issue that this presents to input sampling methods is that the number of states within the local minimum is infinite because of the lack of a discretized state space.

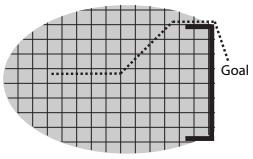

Fig. 4. Illustration of the necessity of an implicit state grid.

The secondary effect resulting from the nature of input sampling as well as the lack of a grid, is that the likelihood of two states being identical is extremely small. All $A^\star$-like algorithms utilize Bellman's optimality principle to improve the path to a particular state by updating the paths through that state when a lower cost alternative is found. This feature is essential to the proper functioning of the algorithm and requires a mechanism to identify when states are close enough to be considered the same. The scenario presented in Fig. 5 is a situation for which the lack of this mechanism would generate an inefficient path. In this situation, vertex v_1 is selected for expansion after which the lowest cost vertex is v_3. The *implicit state grid* then recognizes that v_2 and v_3 are close enough to be considered the same and updates the path to their grid cell to be path c since $c < a + b$.

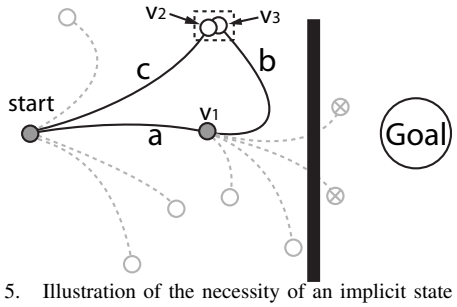

Fig. 5. Illustration of the necessity of an implicit state grid.

The concept of an *implicit state grid* [17] is introduced as a solution to both of these problems. The implicit grid ensures that the graph generated by the SBMPC algorithm is constructed such that only one active state exists in each grid cell, therefore limiting the number of vertices that can exist within any finite region of the state space. It also allows for the efficient storage of potentially infinite grids by only storing the grid cells that contain vertices, which is increasingly important for higher dimensional problems [17]. The resolution of the grid is a significant factor in determining the performance of the algorithm with more fine grids in general requiring more computation time, due to the increased number of states, with the benefit being a more optimal solution. The resolution therefore is a useful tool that enables SBMPC to effectively make the trade off between solution quality and computational performance. However, it must be stated that increasing the resolution significantly beyond the accuracy of the prediction could result in the generation of infeasible solutions.

The implicit state grid can also be restricted to represent the subset of the state space that contains the problem objective, to further reduce the number of states that must be explored. For example, a 2D path planning problem using an Ackerman steered vehicle model could use an implicit state grid that does not contain the vehicle orientation as long as the planning objective is only to reach a desired position. The use of a reduced state grid has been shown to be effective in practice but does invalidate all claims of guaranteed convergence and optimality. The research of [4] develops an input sampling methodology that cleverly eludes the need for state gridding by employing a closed-loop model that has as its inputs, the commanded position in the

configuration space. As mentioned in the Introduction, this restricts their approach to closed-loop systems that have a controller that allows these types of inputs, whereas SBMPC is more generally applicable.

B. SBMPC Variables

SBMPC operates on a dynamic directed graph G which is a set of all vertices and edges currently in the graph. $SUCC(v)$ denotes the set of successors (children) of vertex $v \in G$ while $PRED(v)$ denotes the set of all predecessors (parents) of vertex $v \in G$. The cost of moving from vertex v to vertex $v' \in SUCC(v)$ is denoted by $c(v, v')$, where $0 < c(v, v') < \infty$. The optimization component of SBMPC is called Sampling-Based Model Predictive Optimization (SBMPO) and is an algorithm that determines the shortest path from a given start vertex $v_{start} \in G$ and a given goal vertex $v_{goal} \in G$. The goal distance of vertex $v \in G$ is denoted by $gd(v)$ and is the cost of the shortest path from v to v_{goal}. Similarly, $g^\star(v)$ denotes the start distance of vertex $v \in G$ and is the cost of the shortest path from v_{start} to v. All start distances satisfy

$$g^\star(v) = \begin{cases} 0, & \text{if } v = v_{start} \\ \min_{v' \in PRED(v)}(g^\star(v') + c(v', v)), & \text{otherwise.} \end{cases} \tag{7}$$

Since the start distance is not initially known, SBMPC maintains two estimates of $g^\star(v)$. The first estimate $g(v)$ is essentially the current cost from v_{start} to the vertex v while the second estimate, $rhs(v)$ is a one-step lookahead based on $g(v')$ for $v' \in PRED(v)$ and is thus potentially better informed than $g(v)$ [15]. $Rhs(v)$ satisfies

$$rhs(v) = \begin{cases} 0, & \text{if } v = v_{start} \\ \min_{v' \in PRED(v)}(g(v') + c(v', v)), & \text{otherwise.} \end{cases} \tag{8}$$

A vertex v is locally consistent iff $g(v) = rhs(v)$ and locally inconsistent iff $g(v) \neq rhs(v)$. If all vertices are locally consistent then $g(v)$ satisfies (8) for all $v \in G$ and is therefore equal to the start distance as defined in (7). This enables the ability to trace the shortest path from v_{start} to any vertex v by starting at v and moving to any predecessor v' that minimizes $g(v') + c(v', v)$ until v_{start} is reached.

To facilitate fast re-planning SBMPO does not make every vertex locally consistent after an edge cost change and instead uses a heuristic function $h(v, v_{goal})$ to focus the search so that it only updates $g(v)$ for vertices necessary to obtain the shortest path. The heuristic is used to approximate the goal distances and must follow the triangle inequality: $h(v_{goal}, v_{goal}) = 0$ and $h(v, v_{goal}) \leq c(v, v') + h(v', v_{goal})$ for all vertices $v \in G$ and $v' \in SUCC(s)$. SBMPO uses the heuristic along with the start distance estimates to rank the $OPEN$ list, a priority queue containing the locally inconsistent vertices and thus all the vertices that need to be updated in order to make them locally consistent. The priority of a vertex is determined by a two component key vector:

$$key(v) = \begin{bmatrix} \min(g(v), rhs(v)) + h(v, v_{goal}) \\ \min(g(v), rhs(v)) \end{bmatrix}, \tag{9}$$

where the keys are ordered lexicographically with the smaller key values having a higher priority. This ordering means that the key values of expanded vertices are nondecreasing over time.

C. The SBMPC Algorithm

SBMPC is presented in this paper as a methodology to generate optimal trajectories. However, it can also be used to follow a trajectory as in Receding Horizon MPC. This is accomplished by selecting coincidence points along the trajectory and treating each as a set point.

SBMPC represents a generic framework for the application of various types of models and sampling strategies. These parameters can be chosen such that SBMPC reduces to several existing algorithms. For example, using standard grid sampling and a model for which the inputs are equal to the change in the system outputs, SBMPC reduces to a dynamic variant of the $A^\star$ algorithm, known as $LPA^\star$ [15]. A heuristically-biased RRT [18] can be produced through the use of a closed-loop model with inputs in the system configuration space.

Algorithm 1 is the SBMPC algorithm while Algorithms 2 and 3 respectively describe the SBMPO and neighbor generation functions. The main algorithm follows the general structure of MPC where SBMPO repeatedly computes the shortest path between the current state $x_{current}$ and the goal state x_{goal}. After a single path is generated $x_{current}$ is updated to reflect the implementation of the first computer control input and the graph G is updated to reflect any system changes.

SBMPO repeatedly generates the neighbors of locally inconsistent vertices until v_{goal} is locally consistent and the key of the next vertex in the $OPEN$ list is no smaller than $key(v_{goal})$. The algorithm then deals with two potential cases based on the consistency of the expanded vertex v_{best}. If the vertex is locally overconsistent, $g(v) > rhs(v)$ and $rhs(v) = g^\star(v)$. In this case, letting $g(v) \leftarrow rhs(v)$ makes the vertex locally consistent. The successors of v are then updated. The update vertex proccess includes recalculating $rhs(v)$, checking local consistency and either adding or removing the vertex from $OPEN$ accordingly. For the case when the vertex is locally underconsistent, $g(v) \leftarrow \infty$ makes the vertex either locally consistent or overconsistent. This change can affect the vertex along with it's successors which then go through the vertex update process.

The neighbor generation method generates a set of pseudo-random samples in the input space that are then used to predict a set of paths to a new set of states with the $x_{current}$ being the initial condition. The branching factor, B determines the number of paths that will be generated. The path is represented by a sequence of states $x(t)$ for $t = t_1, t_1 + \Delta t, \cdots, t_2$, where Δt is the model step size. The set of states that do not violate any state or obstacle constraints is called $\mathbf{X}_{free}$. If $x(t) \in \mathbf{X}_{free}$ then the new vertex x_{new} and the connecting edge can be added to the graph. If $x_{new} \in STATE\_GRID$ then the vertex currently exists in the graph and only the new path to get to the existing vertex needs to be added.

Algorithm 1 Sampling-Based Model Predictive Control

```
x_current ⇐ start
repeat
    SBMPO ( )
    Update system state, x_current
    Update graph, G
until the goal state is achieved
```

Algorithm 2 SBMPO ( )

```
while OPEN.TopKey() < v_goal.key || v_goal.rhs ≠ v_goal.g do
    v_best ⇐ OPEN.Top()
    Generate_Neighbors ( v_best, B)
    if v_best.g > v_best.rhs then
        v_best.g = v_best.rhs
        for all s ∈ SUCC_{v_best} do
            Update the vertex, s
        end for
    else
        v_best.g ⇐ ∞
        for all s ∈ SUCC(v_best) ∪ v_best do
            Update the vertex, s
        end for
    end if
end while
```

Algorithm 3 Generate_Neighbors (Vertex v, Branching B)

```
for i = 0 to B do
    Generate sampled input, u ∈ R^u ∩ U_free
    for t = t1 : dt_integ : t2 do
        Evaluate model: x(t) = f(v.x, u)
        if x(t) ∉ X_free(t) then
            Break
        end if
    end for
    x_new = x(t2)
    if x_new ∈ STATE_GRID and x_new ∈ X_free then
        Add Edge(v.x, x_new) to graph, G
    else if x_new ∈ X_free then
        Add Vertex(x_new) to graph, G
        Add Edge(v.x, x_new) to graph, G
    end if
end for
```

D. SBMPC Implementation Details

Due to the fact that SBMPC more closely resembles a goal-directed algorithm than traditional MPC, implementation of the algorithm is a topic that must be addressed. One of the downfalls of SBMPC is that it uses a nonstandard optimization method. Therefore, it is not likely that SBMPC can be easily implemented using readily available optimization tools and involves a great deal of programming effort. The optimization used in our implementation of SBMPC was derived through the generalization of the $LPA^\star$ algorithm so that it can operate on a dynamic graph as opposed to a static grid. In order to facilitate this generalization while compensating for the limited adaptability of current libraries, a complete graph library was developed. It is also necessary to modify the algorithm so that the graph can be generated online by integrating the system model based on the sampled set of control inputs. One of the primary developments that enabled the successful implementation of SBMPC is the implicit state grid. Note that a benefit of the SBMPC method is there are only two tuning parameters, the grid resolution and the number of samples. This factor makes it easy for a layman to implement the SBMPC method in practice.

E. Completeness/Stability

A formal presentation of the completeness of SBMPC will be presented in a future publication. The completeness proof can be inferred by the proofs of the $LPA^\star$ algorithm [15] under two conditions: (1) the graph must be constructed such that a finite number of vertices exists within a finite state space and (2) the implicit state grid must represent the full state space and not simply the states used in the planning objective. This reinforces the necessity of the *Implicit State Grid*, which prevents the input sampling from generating infinitely dense vertices in the output space. (The completeness and optimality properties rely on the connectedness of the grid which is established by the vertex generation methodology.) Completeness analysis is equivalent to stability analysis with more traditional MPC algorithms

IV. SIMULATION RESULTS

The results presented in this section have two purposes: 1) to compare SBMPC with standard MPC implementations on the type of autonomous ground vehicles (AGV) motion planning problem that originally motivated the development of SMBPC, and 2) to demonstrate SBMPC's ability to solve problems outside the domain of robot motion planning. Hence, SBMPC and MPC are first compared on the problem of trajectory generation for an (Ackerman steered) AGV moving in cluttered environments [19], a difficult problem with many local minima. (Whenever the vehicle is behind an obstacle or group of obstacles and has to increase its distance from the goal to achieve the goal, it is in a local minimum position). Finally, SBMPC is compared with MPC using a set point problem for a nonlinear, continuous stirred-tank reactor (CSTR).

A. SBMPC and MPC Motion Planning for an Ackerman Steered Vehicle in Cluttered Environments

Motion planning for an Ackerman steered vehicle is developed using the Euler discretization of the kinematic model corresponding to a model time step T_s, which is given by

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) + T_s \cos\theta(k) u(k) \\ y(k) + T_s \sin\theta(k) u(k) \\ \theta(k) + T_s \frac{tan\delta(k)}{L} u(k) \end{bmatrix}, \quad (10)$$

where the inputs $u(k)$ and $\delta(k)$ are respectively the forward velocity and steering rate, L is the longitudinal wheel separation, and $\delta(k)$ and $\theta(k)$ are respectively the steering and orientation angles. The velocity $u(k)$ was constrained to lie in the interval $(0, 5)$m/s such that the vehicle is only allowed to move forward, and the steering rate $\delta(k)$ was constrained to the interval $(-\frac{\pi}{6}, \frac{\pi}{6})$rad/s.

The basic problem is to use the kinematic model (10) to plan a minimum-distance trajectory for the AGV from a start point to a goal point while avoiding the numerous obstacles of a cluttered environment. MPC was based on a modified cost function,

$$\min_{\{u(k),\ldots,u(k+M-1)\}} J = \|G - y(N)\|^2_{Q_1} + \sum_{i=1}^{N} \|y(k+i) - y(k+i-1)\|^2_{Q_2}, \quad (11)$$

which is very similar to the SBMPC cost function given in (1). However, since the optimization is not goal-directed as for SBMPC, the first term is needed in (11) to force the system to the desired goal.

In order to evaluate SBMPC it is compared with $\mathrm{MPC_{SQ}}$, a nonlinear MPC implementation that employs sequential quadratic programming (SQP) using an active set strategy, and $\mathrm{MPC_{GA}}$, a NMPC implementation which uses a genetic algorithm (GA) to determine the system inputs. Genetic algorithms, though relatively slow, are considered here because $\mathrm{MPC_{SQ}}$ and other MPC implementations built on Newton-type algorithms tend to fail in cluttered environment planning because of convergence to one of the many local minima. Due to the fundamental differences in SBMPC and traditional MPC, two different parameter sets were used for $\mathrm{MPC_{GA}}$ and the corresponding simulations are denoted by $\mathrm{MPC_{GA,1}}$ and $\mathrm{MPC_{GA,2}}$. The parameters for the simulations are shown in Table I. It is assumed that the time step in the SBMPC cost function (1) is T_c while that for the MPC cost function (11) is T_s. For $\mathrm{MPC_{SQ}}$, $\mathrm{MPC_{GA}}$ and SBMPC the constraints were checked every time the model was updated (i.e., with period T_s) and $T_c > T_s$ corresponds to the period over which the control inputs were held constant. The parameters for $\mathrm{MPC_{GA,1}}$ were chosen to enable MPC to most resemble SBMPC. However, $\mathrm{MPC_{GA,2}}$ obtained increased optimality and ability to reach the goal by increasing the control horizon to $M = N/2$ and increasing the number of GA optimization generations from 10 to 30. Table I indicates that SBMPC has a control horizon of N. This is because one optimization cycle of SBMPC allows the control input $u(k)$ to vary for $k = 1, 2, \cdots, N-1$.

There were 100 simulations in which 30 obstacles were randomly generated to produce different scenarios. As a result, in Table I the prediction horizon varies because each random scenario requires a different number of steps to traverse from the start position to the goal position. In each simulation, the vehicle length, L = 1m, and implicit state grid resolution is 0.1m. SBMPC was used to solve the optimization problem (1) - (6) with $Q(i) = I$ and $S(i) = 0$, and yielded the optimal number of steps N^* in addition to the control input sequence. For simulations purposes, the other algorithms used $N = N^*$. Traditional MPC can also include N as an optimization variable to solve an optimization problem similar to (1). However, this has primarily been accomplished using mixed integer linear programming [6], which is applicable only to LMPC. To accomplish this for NMPC requires mixed integer nonlinear programming, a much more complex optimization problem which is not in the scope of this paper.

TABLE I

SIMULATION PARAMETERS

	$\mathrm{MPC_{SQ}}$	$\mathrm{MPC_{GA1}}$	$\mathrm{MPC_{GA2}}$	SBMPC
Model Time Step (T_s)	0.1s	0.1s	0.1s	0.1s
Control Update Period (T_c)	1s	1s	1s	1s
Prediction Horizon (N)	[18,40]	[18,40]	[18,40]	[18,40]
Control Horizon (M)	1	1	N/2	N
No. of Input Samples	N/A	N/A	N/A	10
GA Population Size	N/A	10	10	N/A
GA No. of Generations	N/A	10	30	N/A

TABLE II

SIMULATION RESULTS FOR CONTROL UPDATE COMPUTATIONS

	$\mathrm{MPC_{SQ}}$	$\mathrm{MPC_{GA,1}}$	$\mathrm{MPC_{GA,2}}$	SBMPC
First Update Time	6.02s	1.18s	1.62s	3.51e-02s
Mean CPU Time	2.53e-01s	5.88e-01s	8.52e-01s	$*$
Median CPU Time	2.51e-02s	5.69e-01s	8.17e-01s	$*$
Mean Distance [a]	30.76m	45.93m	32.75m	31.06
Success Rate	13%	73%	94%	100%

[a]Distance only includes successful trajectories.

The results of 100 random simulations on a 3.33 GHz Intel Core 2 Duo PC are shown in Table II. Note that unlike the MPC algorithms, the trajectory generated by SBMPC converged to the goal during the first optimization cycle (corresponding to the control update computation). Hence, no replanning was technically required and was not performed in this study. This accounts for the $*$ in the last column of Table II. However, if replanning were performed in a static environment, the median CPU time would be close to zero and the mean CPU time would be close to the first update time divided by N. SBMPC outperformed $\mathrm{MPC_{SQ}}$, $\mathrm{MPC_{GA,1}}$, and $\mathrm{MPC_{GA,2}}$ in terms of both success rate and computational time. (A more optimized versions of the MPC implementations could reduce the gap in computational times.) However, $\mathrm{MPC_{SQ}}$ resulted in a slightly shorter mean distance for the scenarios that it successfully completed. A representative result from the 100 scenarios is shown in Fig. 6. It is a typical scenario in which both the SBMPC, $\mathrm{MPC_{GA,1}}$, and $\mathrm{MPC_{GA,2}}$ simulations simulations reached the goal, but $\mathrm{MPC_{SQ}}$ converged to a local minimum located behind a cluster of obstacles. Even though $\mathrm{MPC_{GA,1}}$ reached the goal it took a less optimal route than SBMPC or $\mathrm{MPC_{GA,2}}$.

B. SBMPC Set Point Control for a CSTR

As stated previously, SBMPC can be employed in general NMPC problems. In this section SBMPC is used to solve a set point problem for a nonlinear continuous stirred-tank reactor (CSTR). For brevity the details of the CSTR are given in [20]. For this example SBMPC was compared only with $\mathrm{MPC_{SQ}}$, which used a prediction horizon of 30 steps and a control horizon of 3 steps. The SBMPC implementation utilized a branching factor of 20 and an implicit state grid resolution of 0.001mol/liter.

Using a distance cost function based on x_1 and x_2 of the form (7) with $S(i)$=0, SBMPC was able to compute a

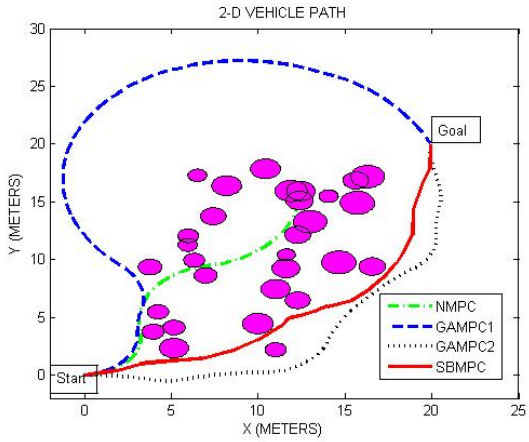

Fig. 6. Typical scenario in which the nonlinear MPC simulation failed to enable the AGV to reach the goal.

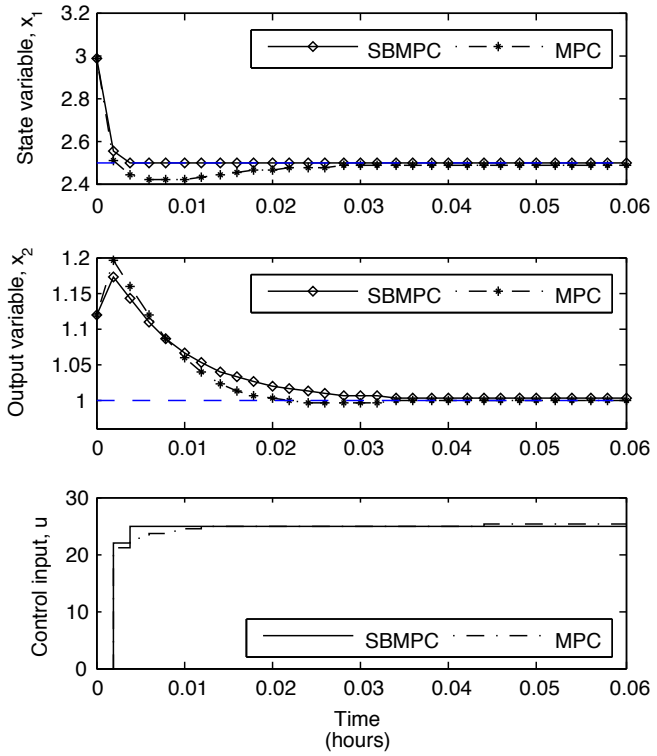

Fig. 7. Isothermal Continuous stirred-tank reactor (CSTR).

solution in 0.008s while $\mathrm{MPC_{SQ}}$, using a cost function of the form (1) with $S(i)$=0, required a CPU time of 0.04s. The simulation results of both are shown in Fig. 6 and illustrate the ability of SBMPC to solve nonlinear set point problems. The $\mathrm{MPC_{SQ}}$ algorithm produced a solution that converged slightly faster in terms of x_2, but resulted in undershoot and a longer convergence time for x_1.

V. CONCLUSION

Sampling-Based Model Predictive Control has been shown to effectively generate a control sequence for a nonlinear system in the presence of a number of nonlinear constraints. SBMPC exploits sampling-based concepts along with the $LPA^{\star}$ incremental optimization algorithm to achieve the goal of being able to quickly determine control updates while avoiding local minima. In addition, due to the nature of SBMPC, robustness is achieved through real time recomputation at every time step.

The SBMPC solution is globally optimal *subject to the chosen sampling method*. This is true whether the model is linear or nonlinear, which is a very powerful feature of the algorithm. When the entire state space is gridded, the SBMPC algorithm guarantees that the algorithm will converge to a solution if one exists. In addition, SBMPC has been applied to a seven degree of freedom robot manipulator. The results for this higher dimension application will be reported in a future publication.

REFERENCES

[1] L. Magni, G. D. Nicolao, and R. Scattolini, "Model predictive control: output feedback and tracking of nonlinear systems," in *Nonlinear Predictive Control: Theory and Practice*, ser. IEE Control Engineering Series 61, B. Kouvaritakis and M. Cannon, Eds. London, United Kingdom: The Institution of Electrical Engineers, 2001, ch. 3, pp. 61–80.

[2] H. Michalska and D. Q. Mayne, "Robust receding horizon control of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 38, no. 1, pp. 1623–1633, November 1993.

[3] D. D. Dunlap, E. G. Collins, Jr., and C. V. Caldwell, "Sampling based model predictive control with application to autonomous vehicle guidance," *Florida Conference on Recent Advances in Robotics*, May 2008.

[4] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transaction on Control Systems Technology*, vol. 17, no. 5, pp. 1105 – 1118, Sep 2009.

[5] W. Yu, O. Chuy, Jr., E. G. Collins, Jr., and P. Hollis, "Dynamic modeling of a skid-steered wheeled vehicle with experimental vertification," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2009.

[6] A. Richards and J. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," *American Control Conference*, pp. 1936–1941, 2002.

[7] D. Shim, J. Kim, and S. Sastry, "Decentralized nonlinear model predictive control of multiple flying robots in dynamic environment," *IEEE Conference on Decision and Control*, 2003.

[8] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[9] S. R. Lindemann and S. M. LaValle, "Incremental low-discrepancy lattice methods for motion planning," *International Conference on Robotics & Automation*, pp. 2920–2927, September 2003.

[10] J. Maciejowski, *Predictive Control with Constraints*. Haslow, UK: Prentice Hall, 2002.

[11] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., 1998.

[12] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics & Automation*, vol. 12, no. 4, pp. 566 – 580, June 1996.

[13] M. Likhachev and A. Stentz, "R* search," *AAAI*, pp. 1–7, Apr 2008.

[14] J. H. Halton, "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals," *Numerishe Mathematik*, 1960.

[15] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, 2004.

[16] J. Nocedal and S. J. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research. New York: Springer, 1999.

[17] C. Ericson, *Real–Time Collision Detection*, D. H. Elberly, Ed. Elsevier, 2005.

[18] C. Urmson and R. Simmons, "Approaches for heuristically biasing rrt growth," in *IEEE/RSJ IROS 2003*, October 2003.

[19] M. F. Selekwa, D. D. Dunlap, and E. G. Collins, Jr., "Implementation of multi-valued fuzzy behavior control for robot navigation in cluttered environments," in *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April 2005 2005, pp. 3699–3706.

[20] E. S. Meadows and J. B. Rawlings, *Nonlinear Process Control*. Prentice-Hall, Inc., 1997, ch. 5, pp. 233 – 310.