11

Motion Planning for Mobile Robots Via Sampling-Based Model Predictive Optimization

Damion D. Dunlap[1], Charmane V. Caldwell[2], Emmanuel G. Collins[2], Jr. and Oscar Chuy[2]

[1]*Naval Surface Warfare Center - Panama City Division*
[2]*Center for Intelligent Systems, Control and Robotics (CISCOR)*
FAMU-FSU College of Engineering
U.S.A

1. Introduction

Path planning is a method that determines a path, consecutive states, between a start state and goal state, LaValle (2006). However, in motion planning that path must be parameterized by time to create a trajectory. Consequently, not only is the path determined, but the time the vehicle moves along the path. To be successful at motion planning, a vehicle model must be incorporated into the trajectory computation. The motivation in utilizing a vehicle model is to provide the opportunity to predict the vehicle's motion resulting from a variety of system inputs. The kinematic model enforces the vehicle kinematic constraints (i.e. turn radius, etc.), on the vehicle that limit the output space (state space). However, the kinematic model is limited because it does not take into account the forces acting on the vehicle. The dynamic model incorporates more useful information about the vehicle's motion than the kinematic model. It describes the feasible control inputs, velocities, acceleration and vehicle/terrain interaction phenomena. Motion planning that will require the vehicle to perform close to its limits (i.e. extreme terrains, frequent acceleration, etc.) will need the dynamic model. Examples of missions that would benefit from using a dynamic model in the planning are time optimal motion planning, energy efficient motion planning and planning in the presence of faults, Yu et al. (2010).

Sampling-based methods represent a type of model based motion planning algorithm. These methods incorporate the system model. There are current sampling-based planners that should be discussed: The Rapidly-Exploring Random Tree (RRT) Planner, Randomized $A^\star$ ($RA^\star$) algorithm, and the Synergistic Combination of Layers of Planning (SyCLoP) multi-layered planning framework. The Rapidly-Exploring Random Tree Planner was one of the first single-query sampling based planners and serves as a foundation upon which many current algorithms are developed. The RRT Planner is very efficient and has been used in many applications including manipulator path planning, Kuffner & LaValle. (2000), and robot trajectory planning, LaValle & Kuffner (2001). However, the RRT Planner has the major drawback of lacking any sort of optimization other than a bias towards exploring the search space. The $RA^\star$ algorithm, which was designed based on the RRT Planner, addresses this drawback by combining the RRT Planner with an $A^\star$ algorithm. The SyCLoP framework is

presented because it not only represents a very current sampling-based planning approach, but the framework is also one of the few algorithms to directly sample the control inputs.
Originally, this research began by applying nonlinear model predictive control (NMPC), implemented with sequential programming, to generate a path for an autonomous underwater vehicle (AUV), Caldwell et al. (2006). As depicted in Fig. 1, NMPC was attractive because it is an online optimal control method that incorporates the system model, optimizes a cost function and includes current and future constraints all in the design process. These benefits made planning with NMPC promising, but there were weaknesses of NMPC that had to be addressed. Since MPC must solve the optimization problem online in real-time, the method was limited to slow systems. Additionally, even though models were used in the design process, linear models where typically used in order to avoid the local minima problem that accompany the use of nonlinear models. In order to exploit the benefits of MPC these issues had to be addressed.

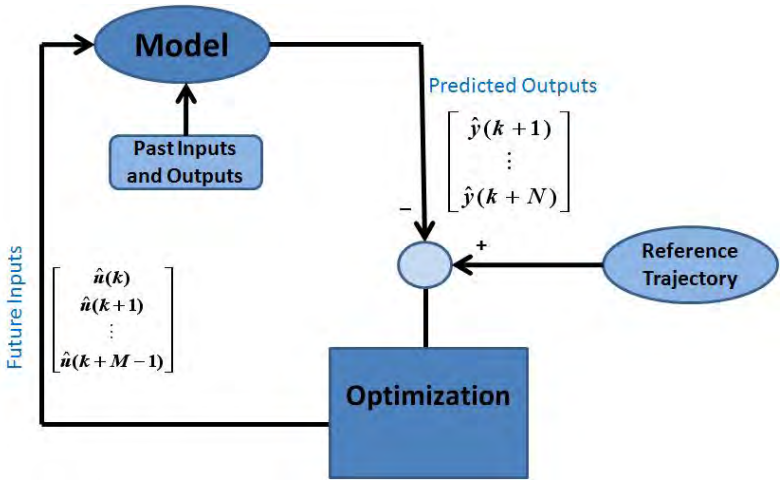


Fig. 1. The stages of the MPC algorithm.

Since the robotics and AI communities had the same goal for planning but have different approaches that tend to yield computationally efficient algorithms, it was decided to integrate these various concepts to produce a new enhanced planner called Sampling Based Model Predictive Control (SBMPC). The concept behind SBMPC was first presented in Dunlap et al. (2008). Instead of utilizing traditional numerical methods in the NMPC optimization phase in Fig. 1, Sampling Based Model Predictive Optimization (SBMPO) uses $A^{\star}$ type optimization from the AI community. This type of graph search algorithm results in paths that do not become stuck in local minima. In addition, the idea of using sampling to consider only a finite number of solutions comes from robotic motion planning community. Sampling is the mechanism used to trade performance for computational efficiency. Instead of sampling in the output space as traditional sampling based planning methods, SBMPC follows the view of traditional MPC and SyCLoP, which samples the input space. Thus, SBMPC draws from the control theory, robotics and AI communities.
Section 2 of this chapter will present the novel SBMPC algorithm in detail and compare Sampling Based Model Predictive Optimization and traditional Sampling based methods. Section 3 provides simulation results utilized on an AUV kinematic model. Section 4 presents

results of both an AUV and an unmanned ground vehicle (UGV) that perform steep hill climbing. An evaluation of SBMPO tuning parameters on the computation time and cost is presented in Section 5. Finally, Section 6 concludes and presents future work.

2. SBMPC algorithm

This section provides the SBMPC Algorithm and the comparison of SBMPO to other traditional sampling based methods. However, first the variables used in the algorithm are described. The SBMPO algorithm and terms follow closely with Lifelong Planning $A^\star$ ($LPA^\star$), Koenig et al. (2004). However, the variation is in the Generate Neighbor algorithm which generates the next state by integrating the system model and considering constraint violations. All the components of SBMPC are described in this Section, but the later simulation results in Section 3 and 4 utilize only the SBMPO and Generate Neighbors algorithms.

2.1 SBMPC variables

SBMPC operates on a dynamic directed graph G which is a set of all nodes and edges currently in the graph. $SUCC(n)$ represents the set of successors (children) of node $n \in G$ while $PRED(n)$ denotes the set of all predecessors (parents) of node $v \in G$. The cost of traversing from node n' to node $n \in SUCC(n')$ is denoted by $c(n', n)$, where $0 < c(n', n) < \infty$. The optimization component is called Sampling Based Model Predictive Optimization and is an algorithm that determines the optimal cost (i.e. shortest path, shortest time, least energy, etc.) from a start node $n_{start} \in G$ to a goal node $n_{goal} \in G$.

The start distance of node $v \in G$ is given by $g^\star(v)$ which is the cost of the optimal path from the given start node v_{start} to the current node v. SBMPC maintains two estimates of $g^\star(v)$. The first estimate $g(v)$ is essentially the current cost from v_{start} to the node v while the second estimate, $rhs(v)$, is a one-step lookahead estimate based on $g(v')$ for $v' \in PRED(v)$ and provides more information than the estimate $g(v)$. The $rhs(v)$ value satisfies

$$rhs(v) = \begin{cases} 0, & \text{if } v = v_{start} \\ \min_{v' \in PRED(v)}(g(v') + c(v', v)), & \text{otherwise.} \end{cases} \tag{1}$$

A node v is locally consistent iff $g(v) = rhs(v)$ and locally inconsistent iff $g(v) \neq rhs(v)$. If all nodes are locally consistent, then $g(v)$ satisfies (1) for all $v \in G$ and is therefore equal to the start distance. This enables the ability to trace the shortest path from v_{start} to any node v by starting at v and traversing to any predecessor v' that minimizes $g(v') + c(v', v)$ until v_{start} is reached.

To facilitate fast re-planning, SBMPC does not make every node 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 nodes necessary to obtain the optimal cost. 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 nodes $v \in G$ and $v' \in SUCC(s)$. SBMPO employs the heuristic function along with the start distance estimates to rank the priority queue containing the locally inconsistent nodes and thus all the nodes that need to be updated in order for them to be locally consistent. The priority of a node is determined by a two component key vector:

$$key(v) = \begin{pmatrix} k_1(v) \\ k_2(v) \end{pmatrix} = \begin{pmatrix} \min(g(v), rhs(v)) + h(v, v_{goal}) \\ \min(g(v), rhs(v)) \end{pmatrix} \tag{2}$$

where the keys are ordered lexicographically with the smaller key values having a higher priority.

2.2 SBMPC algorithm

The SBMPC algorithm is comprised of three primary methods: Sampling Based Model Predictive Control, Sampling Based Model Predictive Optimization and Generate Neighbor. The main SBMPC algorithm follows the general structure of MPC where SBMPC repeatedly computes the optimal 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 control input and the graph G is updated to reflect any system changes. These steps are repeated until the goal state is reached.

The second algorithm SBMPO is the optimization phase of SBMPC that provides the prediction paths. SBMPO repeatedly generates the neighbors of locally inconsistent nodes until v_{goal} is locally consistent or the key of the next node in the priority que is not smaller than $key(v_{goal})$. This follows closely with the ComputeShortestPath algorithm of $LPA^{\star}$ Koenig et al. (2004). The node, v_{best}, with the highest priority (lowest key value) is on top of the priority que. The algorithm then deals with two potential cases based on the consistency of the expanded node v_{best}. If the node is locally overconsistent, $g(v) > rhs(v)$, the g-value is set to $rhs(v)$ making the node locally consistent. The successors of v are then updated. The update node process includes recalculating $rhs(v)$ and key values, checking for local consistency and either adding or removing the node from the priority que accordingly. For the case when the node is locally underconsistent, $g(v) < rhs(v)$, the g-value is set to ∞ making the node either locally consistent or overconsistent. This change can affect the node along with its successors which then go through the node update process.

The Generate Neighbor algorithm determines the successor nodes of the current node. In the input space, a set of quasi-random samples are generated that are then used with a model of the system to predict a set of paths to a new set of outputs (nodes) with $x_{current}$ being the initial condition. The branching factor B (sampling number) determines the number of paths that will be generated and new successor nodes. 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 neighbor node x_{new} and the connecting edge can be added to the directed graph, G. If $x_{new} \in STATE\_GRID$, then the node currently exists in the graph and only the new path to get to the existing node needs to be added.

Algorithm 1 Sampling Based Model Predictive Control

1: $x_{current} \Leftarrow$ **start**
2: **repeat**
3: SBMPO ( )
4: Update system state, $x_{current}$
5: Update graph, **G**
6: **until** the goal state is achieved

2.3 Comparison of SBMPO and traditional sampling based methods

This section discusses the conceptual comparison between SBMPO and traditional Sampling-based methods. Similar to many other planning methods, there have been many variants of the sampling based methods that seek to improve various aspects of their

Algorithm 2 SBMPO ( )

while $PRIORITY.TopKey() < v_{goal}.key \parallel v_{goal}.rhs \neq v_{goal}.g$ **do**
  $v_{best} \Leftarrow PRIORITY.Top()$
  Generate_Neighbors (v_{best}, **B**)
  if $v_{best}.g > v_{best}.rhs$ **then**
    $v_{best}.g = v_{best}.rhs$
    for all $v \in SUCC_{v_{best}}$ **do**
      Update the node, v
    end for
  else
    $v_{best}.g = \infty$
    for all $v \in SUCC(v_{best}) \cup v_{best}$ **do**
      Update the node, v
    end for
  end if
end while

Algorithm 3 Generate_Neighbors (Vertex v, Branching **B**)

for $i = 0$ to **B** **do**
  Generate sampled input, $u \in \mathbb{R}^u \cap \mathbf{U}_{free}$
  for $t = t_1 : dt_{integ} : t_2$ **do**
    Evaluate model: $x(t) = f(v.x, u)$
    if $x(t) \notin \mathbf{X}_{free}(t)$ **then**
      Break
    end if
  end for
  $x_{new} = x(t_2)$
  if $x_{new} \in STATE\_GRID$ and $x_{new} \in \mathbf{X}_{free}$ **then**
    Add $Edge(v.x, x_{new})$ to graph, **G**
  else if $x_{new} \in \mathbf{X}_{free}$ **then**
    Add $Vertex(x_{new})$ to graph, **G**
    Add $Edge(v.x, x_{new})$ to graph, **G**
  end if
end for

performance. It is not possible to cover every variant, but the purpose of this section is to put in perspective how SBMPO is a variant of the traditional sampling-based method.

2.3.1 Traditional sampling based methods

Examples of traditional sampling based motion planning algorithms include RRTs, LaValle (1998), and probability roadmaps,. A common feature of each of these algorithms is they work in the output space of the robot and employ various strategies for generating samples (i.e., random or pseudo-random points). In essence, as shown in Fig. 2, sampling based motion planning methods work by using sampling to construct a tree that connects the root (initial state) with a goal region.

Most online sampling based planning algorithms follow this general framework:

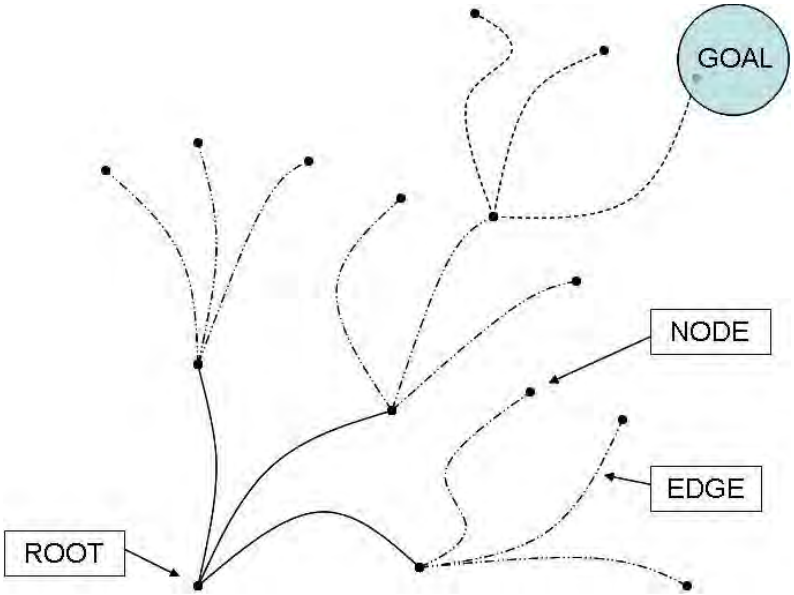


Fig. 2. A tree that connects the root with a goal region.

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 does not 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.

The model is part of the local planning method (LPM), which determines the connection between the newly generated sample and the existing graph. Essentially, it is a two-point value boundary problem.

2.3.2 Similarities of SBMPO and traditional sampling based methods

There are some similarities that both SBMPO and traditional sampling methods share.

2.3.2.1 Sampling

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 community, LaValle (2006). Sampling is the mechanism used to trade performance for computational efficiency. SBMPO employs quasi-random samples of the input space. Properly designed sampling algorithms provide

theoretical assurances 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).

2.3.3 Differences of SBMPO and traditional sampling based methods

Since SBMPO is the outgrowth of both MPC and graph search algorithms, there are some fundamental differences in SBMPO and traditional sampling based methods.

2.3.3.1 Input sampling

There are two primary disadvantages to using output (i.e., configuration space) sampling as is commonly done in traditional sampling based methods. The first limitation lies within the VSM, where the algorithm must determine the most ideal node to expand. This selection is typically made based on the proximity of nodes in the graph to a sampled output node 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. A solution to the problem is to introduce input sampling. The concept of input sampling is not new and has been integrated into methods like the SyCLoP algorithm, Plaku et al. (2010). When the input space is sampled as proposed in this chapter, the need for a nearest-neighbor search is eliminated, and the LPM is reduced to the integration of a system model, and therefore, only generates outputs that are achievable by the system. Sampling the control inputs directly also prevents the need to determine where to connect new samples to the current graph and therefore avoid costly nearest-neighbor searches.

In order to visualize this concept, consider an Ackerman steered vehicle at rest that has position (x, y) and orientation θ, which are the outputs of the kinematic model. The model restricts the attainable outputs. All the dots in Fig. 3 are output nodes obtained from sampling the output space even though only the dots on the mesh surface can physically be obtained by the vehicle. There are a larger number of dots (sampled outputs) in the output space that do not lie in the achievable region (mesh surface). This means those sampled outputs are not physically possible, so traditional sample based methods would have to start the search over. This leads to an inefficient search that can substantially increase the computational time of the planner. The intersection of the grid lines in Fig. 3 correspond to the points in output space generated by a uniform sampling of the model inputs, the left and right wheel velocities. In essence, sampling in the input space leads to more efficient results since each of the corresponding dots in the output space is allowed by the model.

2.3.3.2 Implicit state grid

Although input sampling avoids two of the primary computational bottle-necks of sampling-based motion planning, there is also a downside of input sampling. Input sampling has not been used in most planning research, because it is seen as being inefficient. This type of sampling can result in highly dense samples in the output space since input sampling does not inherently lead to a uniformly discretized output 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 concave 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 Likhachev & Stentz (2008), which discusses that

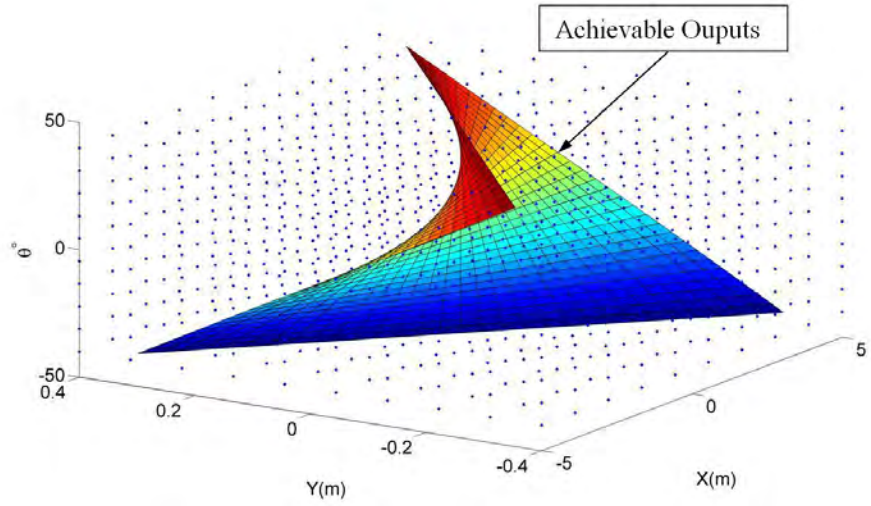


Fig. 3. Illustration of the potential inefficiency of sampling in the output space.

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 output space.

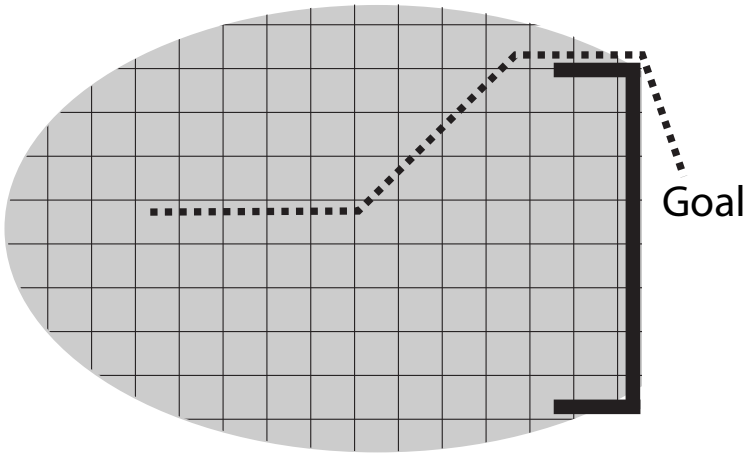


Fig. 4. Illustration of the necessity of an implicit state grid.

The second challenge resulting from the nature of input sampling as well as the lack of a grid is that the likelihood of two outputs (states) being identical is extremely small. All $A^\star$-like

algorithms utilize Bellman's optimality principle to improve the path to a particular output by updating the paths through that output 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 outputs (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, node v_1 is selected for expansion after which the lowest cost node 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 $c < a + b$.

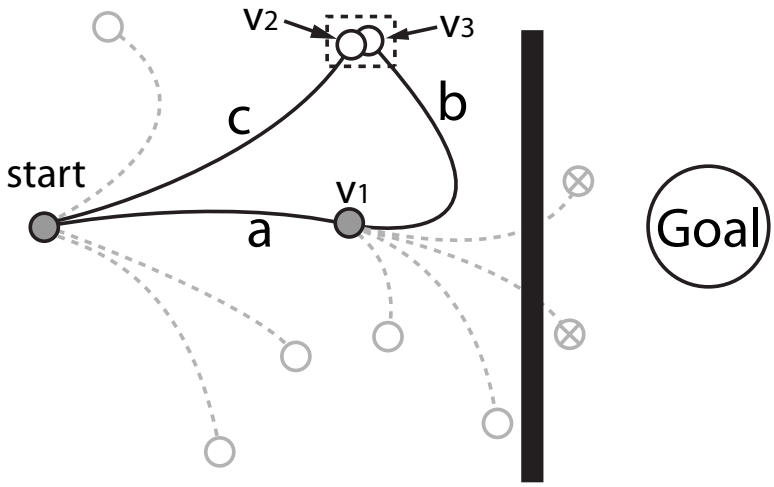


Fig. 5. Illustration of the necessity of an implicit state grid.

The concept of an *implicit state grid*, Ericson (2005), is introduced as a solution to both of the challenges generated by input sampling. The implicit grid ensures that the graph generated by the SBMPC algorithm is constructed such that only one active output (state) exists in each grid cell, limiting the number of nodes that can exist within any finite region of the output space. In essence, the *implicit state grid* provides a discretized output space. It also allows for the efficient storage of potentially infinite grids by only storing the grid cells that contain nodes, which is increasingly important for higher dimensional problems, Ericson (2005). 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 outputs, with the benefit being a more optimal solution. Therefore, the grid resolution is a useful tuning tool that enables SBMPC to effectively make the trade off between solution quality and computational performance.

2.3.3.3 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 study 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 Koenig et al. (2004); LaValle (2006). 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 Nocedal & Wright (1999), which "relaxes" continuous optimization problems, the goal-directed optimization methods are inherently discrete, and have often been used for real-time path planning.
Generally, sampling based methods such as RRTs do not incorporate any optimization and terminate when an initial feasible solution is determined. In essence, instead of determining an optimal trajectory, traditional sampling based methods only attempt to find feasible trajectories. To remedy these problems, the Randomized $A^\star$ ($RA^\star$) algorithm was introduced in Diankov & Kuffner. (2007), as a hybrid between RRTs and the $A^\star$ search algorithm. Similar to $RA^\star$, SBMPO incorporates a goal directed optimization to ensure the trajectory is optimal subject to the sampling.
Although not commonly recognized, goal-directed optimization approaches are capable of solving control theory problems for which the ultimate objective is to plan an optimal trajectory and control inputs to reach a goal (or set point) while optimizing a cost function. Hence, graph search algorithms can be applied to terminal constraint optimization problems and set point control problems. To observe this, consider the tree graph of Fig. 2. 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.

3. 3D motion planning with kinematic model

In order to demonstrate SBMPO capabilities, two local minima scenarios will be considered: 1) a concave obstacle and 2) a highly cluttered area. The purpose is to test how SBMPO handles these types of local minima environments. In this section, the kinematic model of an AUV is used for the motion planning simulations.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi - s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi - c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi s\theta & c\phi s\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \tag{3}$$

where u, v, w are linear velocities in the local body fixed frame along the x, y, z axes, respectively and p, q, r are the angular velocities in the local body fixed frame along the x, y, z axes, respectively. The AUV posture can be defined by six coordinates, three representing the position $x_1 = (x, y, z)^T$ and three corresponding to the orientation $x_2 = (\phi, \theta, \psi)^T$, all with respect to the world frame. The constraints for the vehicle is given in Table 1.
The basic problem in each of these scenarios is to use the kinematic model to plan a minimum distance trajectory for the AUV from a start posture to a goal position while avoiding the obstacles. A 2.93 GHz Intel Core 2 Duo desktop was used for simulations in this Section.

3.1 AUV concave obstacle

As previously stated, SBMPO can handle local minimum problems that other path planning methods have difficulties handling. A local minima problem is a possible scenario a vehicle

Inputs	min	max	States	min	max
u	0 m/s	2 m/s	x	-5 m	30 m
v	-0.1 m/s	0.1 m/s	y	-5 m	30 m
w	-0.1 m/s	0.1 m/s	z	-20 m	0 m
p	$-5^\circ/s$	$5^\circ/s$	ϕ	-15°	15°
q	$-5^\circ/s$	$5^\circ/s$	θ	-15°	15°
r	$-15^\circ/s$	$15^\circ/s$	ψ	-360°	360°

Table 1. Simulation Constraints for the 3D Kinematic Model

can be presented with that has a set of concave obstacles in front of the goal. Note that whenever a 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.
The simulations were run with a sampling number of 25 and grid resolution of 0.1m. The vehicle has a start posture of $(5m, 0m, -10m, 0^\circ)$ and a goal position of $(5m, 10m, -10m)$. As shown in Fig. 6, SBMPO does not get stuck behind the obstacles, but successfully determines a trajectory in 0.59s. The successful traversal is largely due to the optimization method used in SBMPO. The goal-directed optimization allows a more promising node (lower cost) to replace a higher cost node as shown in the the example in Fig 7. Goal-directed optimization can accomplish this because they compute each predicted control input separately 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 optimal predicted control 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. In addition, these results show that SBMPO's implementation of the implicit state grid helps prevent the issues with input sampling discussed in Section 2.3.3.2. Since the implicit grid is applied, it does not require significant time to explore the area around the concave obstacles.

3.2 AUV cluttered multiple obstacles

In Section 3.1, there was one local minimum in the scenario. However, some underwater environments will require the AUV to navigate around multiple cluttered obstacles. This can produce a more complex situation because now there are multiple local minima . The simulations in this section assume there were random start, goal and obstacle locations in order to represent 100 different multiple obstacle underwater environment configurations. The start locations X, Y, Z and ψ were chosen randomly in the respective ranges $[0\ 20]m$, $[0\ 1]m$, $[-12\ -8]m$, $[30^\circ\ 150^\circ]$, and the goal was chosen randomly in the respective ranges $[0\ 20]m$, $[19\ 20]m$, $[-12\ -8]m$. In addition, there were 40 randomly generated obstacles. The 100 simulation runs had a sampling number of 25. Fig. 8 exemplifies one random scenarios generated. In the scenarios SBMPO was capable of allowing the AUV to maneuver in the cluttered environment successfully reaching the goal.
For a vehicle to be truly autonomous, it must be capable of determining a trajectory that will allow it to successfully reach the goal without colliding with an obstacle. In these simulations SBMPO was 100% successful in assuring the vehicle accomplished this task. It is important to consider both SBMPO's mean computation time of $0.43s$ and median computation time of $0.15s$ to compute these trajectories. Since the scenarios generated were random, there were a few scenarios created that were more cluttered which caused a larger CPU time. This is the reason for the discrepancy between the mean and median computation times.

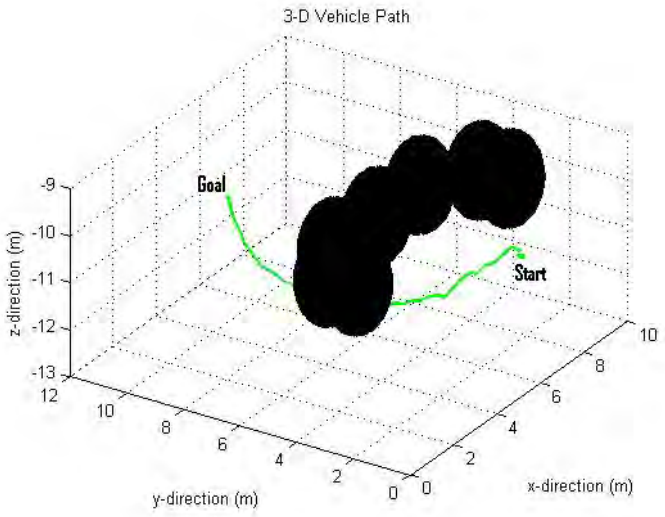


Fig. 6. A local minima scenario.

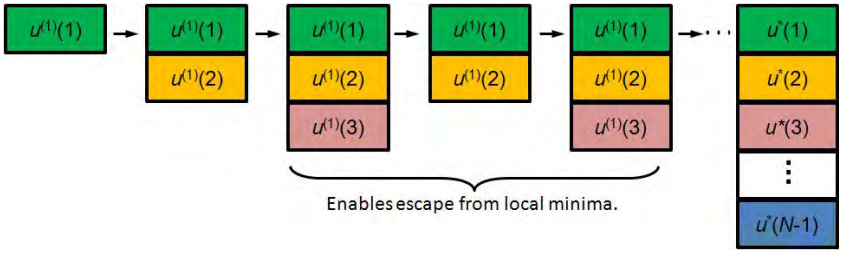


Fig. 7. Example of goal-directed optimization.

4. 3D motion planning with dynamic model

In some scenarios it is sufficient to plan using the kinematic model. However, in cases where the vehicle is pushed to an extreme, it is necessary to consider the vehicles dynamic model when planning.

4.1 Steep hill climbing

In this section, two different types of vehicles, an AUV and an UGV, consider steep hill motion planning using their respective dynamic model. The vehicle must be capable of acquiring a certain amount of momentum to successfully traverse the hill. In order to determine if the vehicle can produce the correct amount of momentum, a dynamic model is not physically capable of traversing.

4.1.1 AUV

The AUV dynamic model used for these simulations can be found in Healey & Lienard (1993). The model constraints are given in Table 2. The SBMPO parameters are in Table 3. The steep

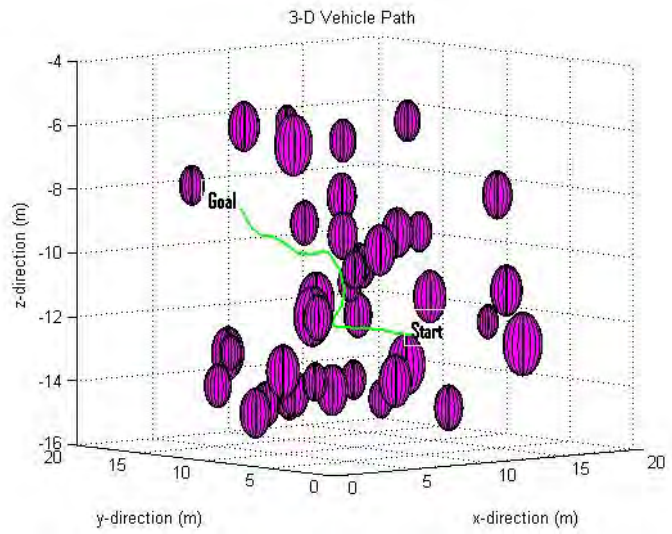


Fig. 8. A random start, goal and obstacle scenario.

hill was constructed utilizing 6 sphere obstacles constraints stacked to give a peak at $8m$. The AUV's start location was $(-4m\ 17m\ -18m\ 0^\circ\ 0^\circ\ 0^\circ)$ and goal was $(19m\ 7m\ -14m)$.

States	min	max	States	min	max	Inputs	min	max
x	-30 m	130 m	u	0 m/s	2 m/s	δ_r	-22°	22°
y	-30 m/s	130 m	v	-2 m/s	2 m/s	δ_s	-22°	22°
z	-20 m	5 m	w	-2 m/s	2 m/s	δ_b	-22°	22°
ϕ	-15°	15°	p	$-5^\circ/s$	$5^\circ/s$	δ_{bp}	-22°	22°
θ	-85°	85°	q	$-5^\circ/s$	$5^\circ/s$	δ_{bs}	-22°	22°
ψ	-360°	360°	r	$-15^\circ/s$	$15^\circ/s$	n	0 rpm	1500 rpm

Table 2. The simulation constraints for the AUV dynamic model.

Model Time Steps	0.5s
Control updates	10s
No. of Input Samples	20
Grid Resolution	0.5

Table 3. The simulation parameters for the AUV dynamic model.

A dynamic model was utilized to determine the path over a steep hill. The AUV was not able to determine a path because the vehicle starts too close to the steep hill to gain momentum. As depicted in Fig. 9, the dynamic model was able to predict that there was not enough momentum to overcome the hill in such a short distance. Note the vehicle constraints do not allow this type of AUV to have a negative velocity which would allow the vehicle to be able to reverse in order to acquire enough momentum. As a result of the vehicle constraint, Fig 9 shows the unsuccessful path. It is not the SBMPO algorithm that cannot successfully

determine a path, but the vehicle constraint (dynamic model) that predicts there was not enough momentum to overcome the hill in such a short distance. In order to demonstrate this consider the same scenario using the kinematic model in Fig 10. SBMPO does determine a path, but this is only because the kinematic model utilized does not provide all the vehicle information to correctly predict the vehicle motion. This further shows the importance of using the proper model when motion planning. The trajectory determined by the planner is only as accurate as the model used.

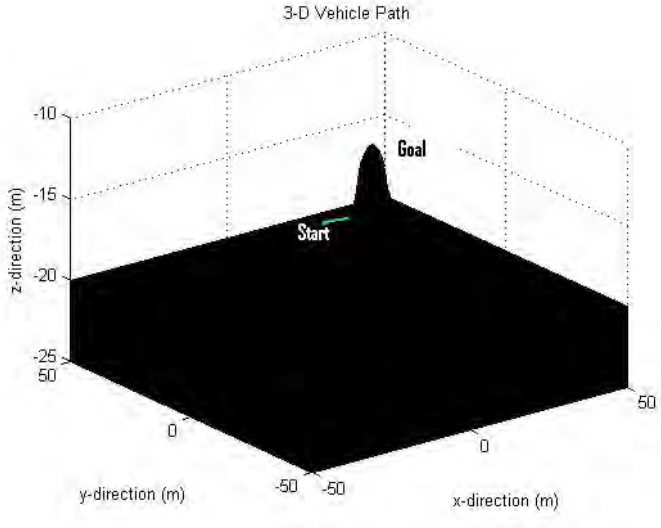


Fig. 9. The AUV dynamic model steep hill scenario.

4.1.2 UGV

This section discusses momentum-based motion planning applied to UGV steep hill climbing. Note that steep hill climbing capability of UGVs is very important to aid the completion of assigned tasks or missions. As an additional requirement, the motion planning is constrained such that the UGV has a zero velocity at the goal (e.g. top of the hill) and this has a unique application, such as reconnaissance, where UGV needs to climb and stop at the top of the hill to gather information. In this section, the momentum-based motion planning is implemented using SBMPO with UGV's dynamic model and a minimum time cost function. The minimum time cost function is employed to achieve zero velocity at the goal.

Figure 11 shows a scenario where a UGV is at the bottom of a steep hill and the task is to climb to the top of the hill. The general approach is to rush to the top of the hill. However, if the torque of the UGV and the momentum are not enough, it is highly possible that the UGV will fail to climb as shown in Fig. 11(a). An alternative approach for the UGV is to back up to gain more momentum and rush to the top of the hill as shown in Fig 11(b). The aforementioned approaches can be done using SBMPO with UGV's dynamic model. SBMPO can generate a trajectory for successful steep hill climbing, and it can also determine if the UGV needs to back up or how far the UGV needs to back up to successfully climb the hill.

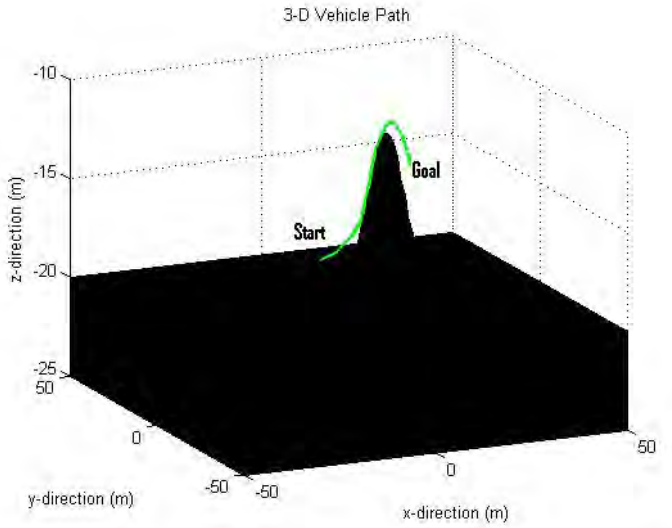
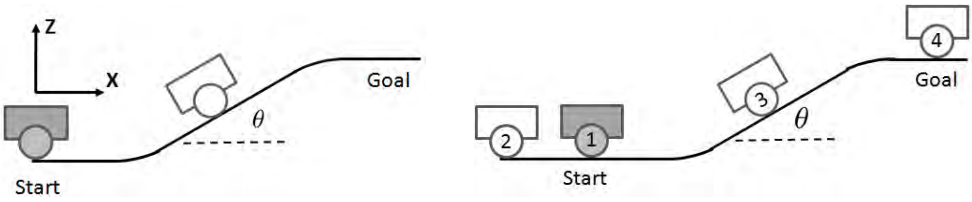


Fig. 10. The AUV kinematic model steep hill scenario.



(a) (b)

Fig. 11. A steep hill climbing scenario for a UGV (a) the UGV rushes to the top of the hill without enough momentum and torque and leads to unsuccessful climb (b) the UGV backs up to gain momentum and leads to a successful climb.

A minimum time cost function is used to implement steep hill climbing with zero velocity at the goal. To formulate the minimum time, consider a system described by

$$\ddot{q} = u; \;\; q(0) = q_0, \;\; \dot{q}(0) = \omega_0, \tag{4}$$

where u is bounded by $-a \leq u \leq b$. The state space description of (4) is given by

$$\dot{q}_1 = q_2, \;\; \dot{q}_2 = u; \; q_1(0) = q_0 \stackrel{\Delta}{=} q_{1,0}, \; q_2(0) = \omega_0 \stackrel{\Delta}{=} q_{2,0}, \tag{5}$$

where $q_1 = q$ and $q_2 = \dot{q}$. It is desired to find the minimum time needed to transfer the system from the original state $(q_{1,0}, q_{2,0})$ to the final state $(q_{1,f}, 0)$, where $q_{1,f} \stackrel{\Delta}{=} q_f$. Since the solution for transferring the system from $(q_{1,0}, q_{2,0})$ to the origin $(0, 0)$ is easily extended to the more general case by a simple change of variable, for ease of exposition it is assumed that

$$q_{1,f} = 0. \tag{6}$$

The minimum time control problem described above can be solved by forming the Hamiltonian and applying the "Minimum Principle" (often referred to as "Pontryagin's Maximum Principle") as described in Bryson & Ho (1975). In fact, the above problem is solved in Bryson & Ho (1975) for the case when the parameters a and b are given by $a = b = 1$. Generalizing these results yields that the minimum time is the solution $\underline{t}_f$ of

$$\begin{aligned} \underline{t}_f^2 - \tfrac{2q_{2,0}}{a}\underline{t}_f &= \tfrac{q_{2,0}^2+2(a+b)q_{1,0}}{ab}, \text{ if } q_{1,0} + \tfrac{q_{2,0}|q_{2,0}|}{2b} < 0, \\ \underline{t}_f^2 + \tfrac{2q_{2,0}}{b}\underline{t}_f &= \tfrac{q_{2,0}^2-2(a+b)q_{1,0}}{ab}, \text{ if } q_{1,0} + \tfrac{q_{2,0}|q_{2,0}|}{2a} > 0. \end{aligned} \tag{7}$$

The minimum time ($\underline{t}_f$) computed using (7) corresponds to a "bang-bang" optimal controller illustrated by Fig. 12, which shows switching curves that take the system to the origin using either the minimum or maximum control input (i.e., $u = -a$ or $u = b$). Depending on the initial conditions, the system uses either the minimum or maximum control input to take the system to the appropriate switching curve. For example, if $(q_{1,0}, q_{2,0})$ corresponds to point p_1 in Fig. 12, then the control input should be $u = -a$ until the system reaches point p_2 on the switching curve corresponding to $u = b$. At this point the control is switched to $u = b$, which will take the system to the origin.

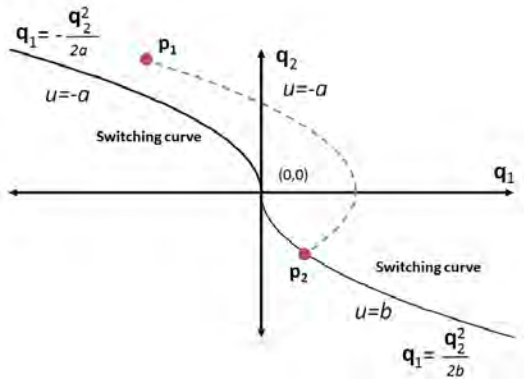


Fig. 12. Illustration of bang-bang minimum time optimal control which yields the minimum time solution $\underline{t}_f$ of (7).

To demonstrate steep hill climbing, the UGV starts at (0,0,0)[m] and the goal is located at (2.5,0,0.76)[m]. As shown in Fig. 13, the hill is described with the following parameters: $R = 1m$, $l = 0.75m$, $d = 0.4m$ and $\theta = 30^o$. A UGV dynamic model discussed in Yu et al. (2010) is used and it is given by

$$M\ddot{q} + C(\dot{q}, q) + G(q) = \tau, \tag{8}$$

where

$$-\tau_{max} < \tau < \tau_{max} \tag{9}$$

and $\tau_{max} = 10Nm$. $\ddot{q}$, $\dot{q}$, and q are respectively the wheel angular acceleration, velocity, and position, M is the inertia, $C(\dot{q}, q)$ is the friction term, and $G(q)$ is the gravity term. Based on the parameters of the hill and the UGV, the maximum required torque to climb quasi-statically the hill is 14.95Nm. This clearly shows that the UGV cannot climb without using momentum.

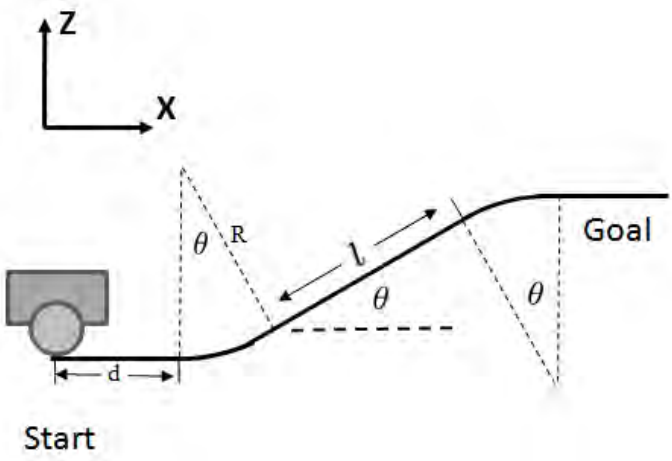


Fig. 13. The UGV steep hill parameters.

The results of the motion planning using SBMPO with the UGV's dynamic model and minimum time cost function are shown in Fig. 14. Fig. 14(a) shows the desired X-Z position of the UGV and Figs. 14(b)-(d) show respectively the desired wheel angular position, velocity, and acceleration, which are the trajectory components of the UGV's. In practice, the resulting trajectory is fed to the UGV's low-level controller for tracking. In Fig. 14(b), the desired wheel angular position starts at zero, and it goes negative (UGV backs up) before it proceeds to the goal. Fig. 14(c) shows the desired angular velocity of the wheel, and it is negative before the UGV accelerates to climb the hill. It also shows that the angular velocity at the goal is zero. The results clearly show that the UGV backs up to increase momentum, which is automatically done by SBMPO.

5. Tuning parameters

Similar to other algorithms, SBMPO has parameters that have to be tuned to guarantee optimal results. SBMPO has two main tuning parameters, the sampling number (branching factor) and grid resolution (size). Each tuning parameter has an effect on the computation time and cost. In this Section, one of the random scenarios from Section 3.2 was investigated.

5.1 Sampling number

The sample number is the number of samples that are selected to span the input space. In order to determine how the sample number effects the computation time of SBMPO the grid resolution was held constant at 0.1m, and the sample number was varied from 10 to 40 by increments of 2. Fig. 15 shows that the effect of the sampling number is nonlinear, so there is no direct relationship between the sample number and computation time. Originally it was thought that an increase in sample number would cause an increase in computation time because there would be more nodes to evaluate. However, as shown in Fig. 15 this is not completely true.

The reason for the nonlinear trend is threefold. First as shown in Fig. 15 by samples 10 and 12 , when there are not enough samples (the sample number is too low) to span the space, it can

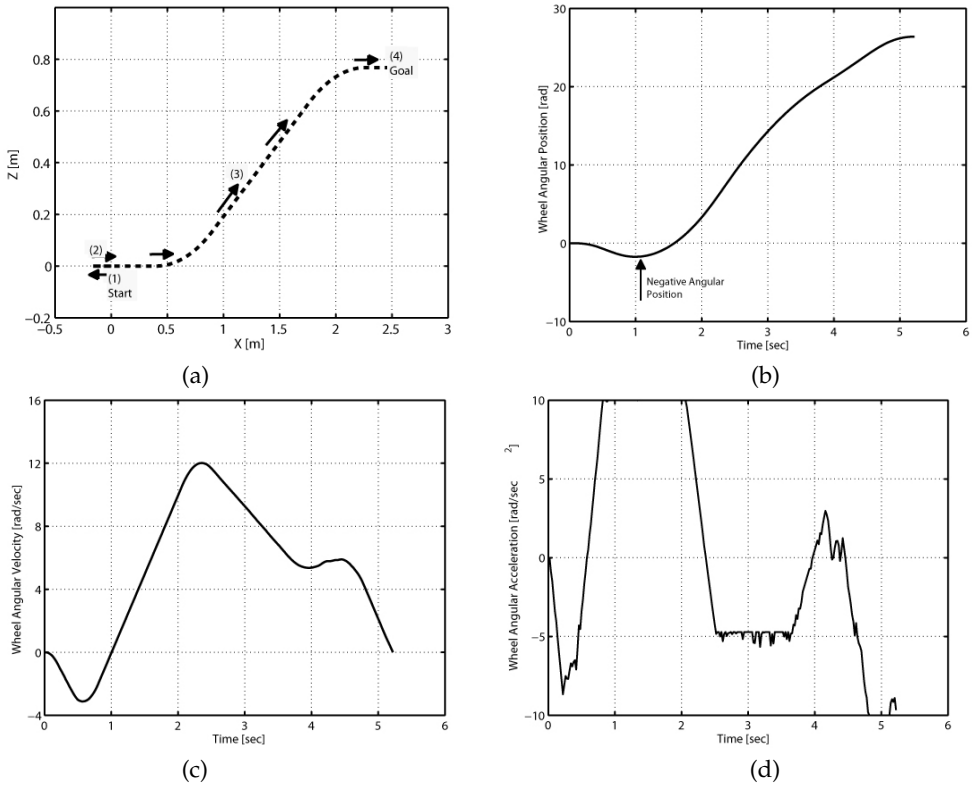


Fig. 14. (a) X-Z position of the UGV (b) wheel angular position (c) wheel angular velocity (d) wheel angular acceleration.

also increase the CPU time, because it takes more iterations (i.e. steps in SBMPO) to determine the solution. A good tuning of the parameter occurs at 14 samples which results in a smaller computation time. The second trend, as shown in Fig. 15 between samples 14 and 22, is that after a good tuning of the parameter, increasing the number of samples also increases the computation times which corresponds to the original hypothesis that an increase in sample number will result in an increase in CPU time. Lastly, a factor that contributes to the effect of the sample number on the computation time is the path produced by SBMPO. It is possible for a larger sample number to have a lower computation time when the path SBMPO generates to the goal encounters a smaller cluster of obstacles. Figs. 16a and 16b show the paths generated respectively using 26 and 36 samples. The path of Fig. 16a which has the lower sampling number takes the AUV through a cluster of obstacles, whereas the path of Fig. 16b which has the larger sample number takes a path that largely avoids the obstacles. Even though Fig. 16b corresponds to a sample number of 36, referring to Fig. 15, its computation time of 0.29s is smaller than that for Fig. 16a, which corresponds to a sample number of 26 and has a computation time of 0.5s.

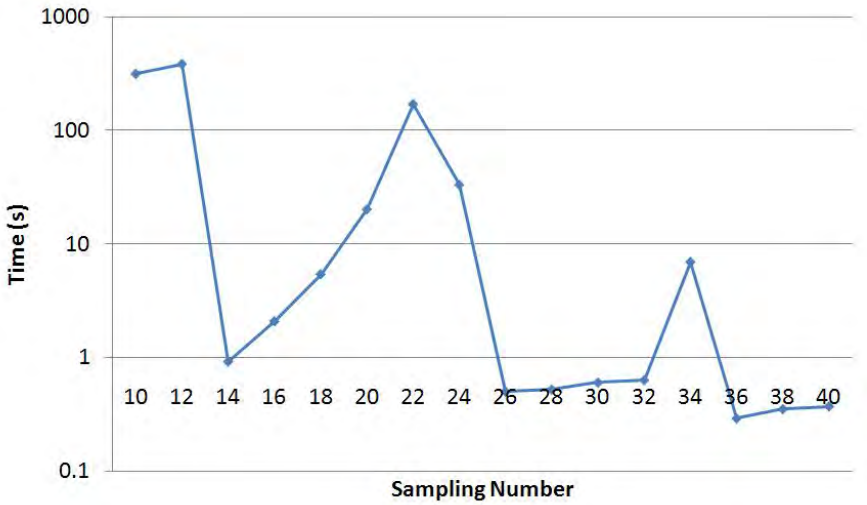
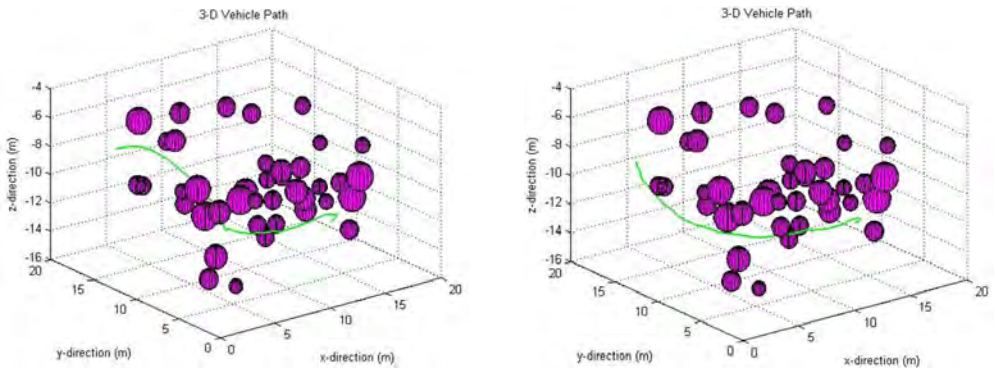


Fig. 15. The effect of sample size on computation time.



(a) (b)

Fig. 16. (a) Scenario with sample number = 26, (b) Scenario with sample number = 36.

Fig. 17 depicts how varying the sample number effects the cost (i.e. distance). The cost is larger in the smaller sample numbers 10 and 12. Afterwards, the variation in the cost is small, which leads to more of an optimal solution.

5.2 Grid size

The grid size is the resolution of the implicit state grid. To evaluate how this tuning parameter effects the computation time, the sampling number was held constant at 25, and the grid resolution was varied between 0.02 to 0.5. Again this tuning parameter is not monotonic with respect to the computation time as depicted in Fig. 18. This shows the importance of properly tuning the algorithm. It may be thought that increasing the grid size would cause

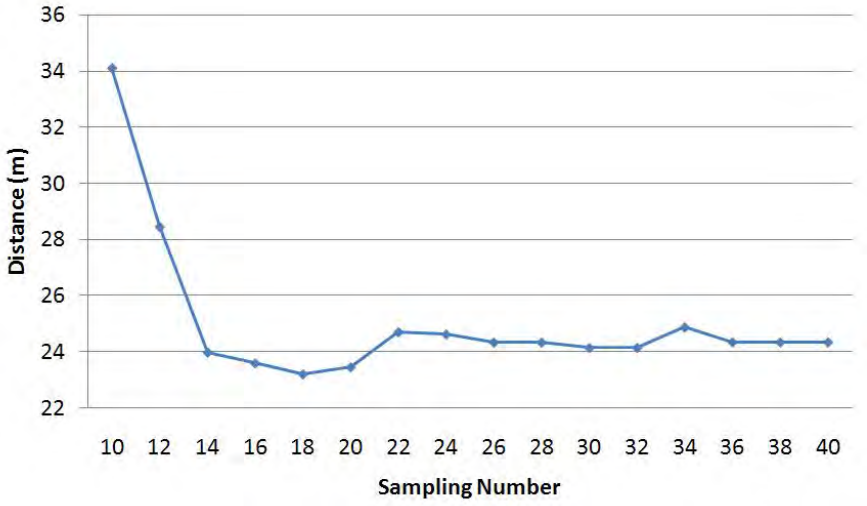


Fig. 17. The effect of sample size on path cost.

less computation. However, the opposite is true. The larger the grid size, the higher the possibility that two nodes are considered as the same state, which leads to the need for more sampling of the input space and an increased computation time. When choosing the grid resolution, it is important to recognize that increasing the grid size tends to lead to higher cost solutions as depicted in Fig. 19.

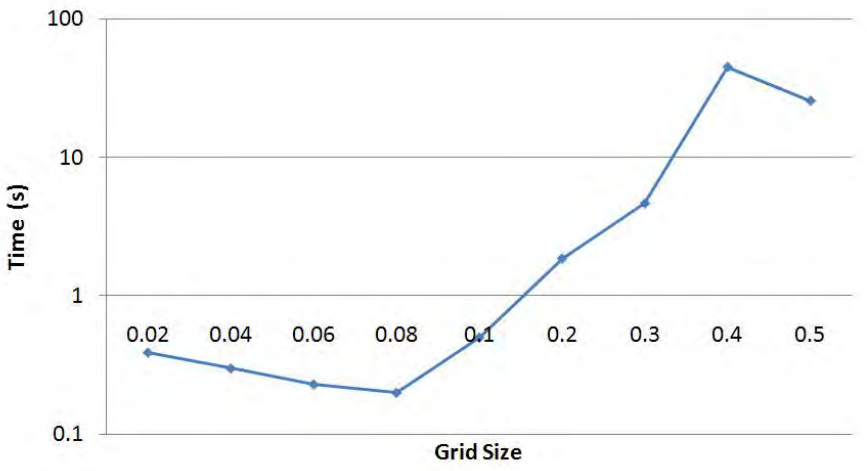


Fig. 18. The effect of grid size on computation time.

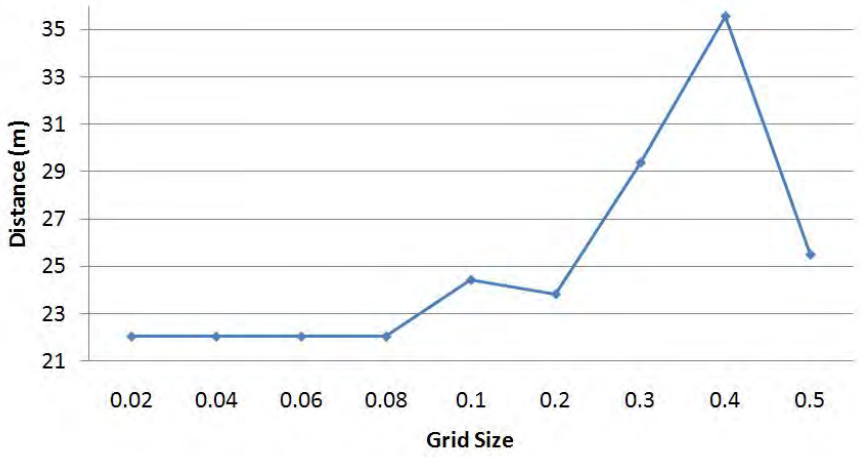


Fig. 19. The effect of grid size on path cost.

6. Conclusion

SBMPO is a NMPC method that exploits sampling-based concepts from the robotics literature along with the *LPA*⋆ incremental optimization algorithm from the AI literature to achieve the goal of quickly and simultaneously determining the control updates and paths while avoiding local minima. The SBMPO solution is globally optimal *subject to the sampling*. Sampling Based Model Predictive Optimization has been shown to effectively generate paths in the presence of nonlinear constraints and when vehicles are pushed to extreme limits. It was determined that SBMPO is only as good as the model supplied to predict the vehicle's movement. Selecting the correct model is important.

The future work is to develop the replanning feature of SBMPC. Currently, SBMPO is applied, but the ability to replan is essential to the SBMPC algorithm. SBMPC utilizes *LPA*⋆ which allows quick replanning of the path without having to completely restart the planning process when new information is obtained or changes in the environment occur. Only the nodes that are affected by a change in the environment must be reevaluated. This reduces the computation time and aids the method in achieving fast computation times. Once the replanning feature of SBMPC is in place, scenarios that include disturbance, model mismatch, unknown obstacles and moving obstacles can be examined to test more realistic situations. The algorithm will also be in a framework that is more comparable to traditional NMPC that only takes the first input and replans at every time step to create a more robust controller. Then SBMPC can be considered a general fast NMPC method that is useful for any nonlinear system or systems subject to nonlinear constraints.

7. References

Bryson, A. & Ho, Y. (1975). *Applied Optimal Control Optimization, Estimation, and Control*, HPC, New York.

Caldwell, C., Collins, E. & Palanki, S. (2006). Integrated guidance and control of AUVs using shrinking horizon model predictive control, *OCEANS Conference* .

Diankov, R. & Kuffner., J. (2007). Randomized statistical path planning, *Conference on Intelligent Robots and Systems* .

Dunlap, D. D., E. G. Collins, J. & Caldwell, C. V. (2008). Sampling based model predictive control with application to autonomous vehicle guidance, *Florida Conference on Recent Advances in Robotics* .

Ericson, C. (2005). *Real–Time Collision Detection*, Elsevier.

Healey, A. & Lienard, D. (1993). Multivariable sliding-mode control for autonomous diving and steering for unmanned underwater vehicle, *IEEE Journal of Oceanic Engineering* 18(3): 327–338.

Koenig, S., Likhachev, M. & Furcy, D. (2004). Lifelong planning A*, *Artificial Intelligence* .

Kuffner, J. J. & LaValle., S. M. (2000). Rrt-connect: An efficient approach to single-query path planning, *IEEE International Conference on Robotics and Automation* p. 995ï£¡1001.

LaValle, S. (1998). Rapidly-exploring random trees: A new tool for path planning, *Technical report*, Iowa State University.

LaValle, S. M. (2006). *Planning Algorithms*, Cambridge University Press.

LaValle, S. M. & Kuffner, J. J. (2001). Randomized kinodynamic planning, *International Journal of Robotics Research* 20(8): 378–400.

Likhachev, M. & Stentz, A. (2008). R* search, *Proceedings of the National Conference on Artificial Intelligence (AAAI)* pp. 1–7.

Nocedal, J. & Wright, S. (1999). *Numerical Optimization*, Springer, New York.

Plaku, E., Kavraki, L. & Vardi, M. (2010). Motion planning with dynamics by synergistic combination of layers of planning, *IEEE Transaction on Robotics* pp. 469–482.

Yu, W., Jr., O. C., Jr., E. C. & Hollis, P. (2010). Analysis and experimental verification for dynamic modeling of a skid-steered wheeled vehicle, *IEEE Transactions on Robotics* pp. 340 – 353.