Using Dynamics to Consider Torque Constraints in Manipulator Planning with Heavy Loads

Oscar Y. Chuy Jr.
Department of Electrical and Computer Engineering
University of West Florida
Pensacola, Florida 32514
Email: ochuy@uwf.edu

Emmanuel G. Collins Jr.
Department of Mechanical Engineering
Florida State University
Tallahassee, Florida 32310
Email: ecollins@eng.fsu.edu

Aneesh Sharma
Department of Mechanical Engineering
Florida State University
Tallahassee, Florida 32310
Email: as10ae@my.fsu.edu

Ryan Kopinsky
Department of Mechanical Engineering
FAMU-FSU College of Engineering
Tallahassee, FL 32310
Email: rkopinsky@fsu.edu

ABSTRACT

Input constraints are active in robot trajectory planning when a mobile robot traverses mobility challenges such as steep hills that limit the acceleration of the robot due to the torque constraints of the motor or engine or in manipulator lifting tasks when the load is sufficiently heavy that the torque constraints of the robot’s motor prevent it from statically supporting the load in regions of the robot’s workspace. This paper presents a general methodology for solving these planning tasks using a minimum time cost function and applies it to the problem of a multiple degree of freedom manipulator lifting a heavy load. Planning for these types of problems requires use of the robot’s dynamic model. Here, we plan using Sampling-Based Model Predictive Optimization, which is only practical if the planning can be done quickly. Hence, substantial attention is given to efficient computations by 1) using the dynamic model without integrating it, 2) using optimal control theory to develop an “optimistic A* estimate of cost-to-goal”, which is in this case a rigorous lower bound on the minimum time from a current state to a goal state, and 3) using prior experience to speed up the computation of a new trajectory. The methodology is experimentally verified for heavy lifting using a two-link manipulator.

Keywords: Motion Planning, Manipulators, Heavy Loads, Torque Constraints

1 Introduction

Input constraints due to actuator authority limitations are inherent in any real-world dynamical system such as a mobile or manipulator robot. Hence, it is possible to generate trajectories that cannot be followed due to these input constraints. For example, this can easily occur when a manipulator is asked to lift a load greater than its rated capacity [1], i.e., the load that it can support statically at any point in its workspace. This can also occur when a mobile robot is asked to traverse a mobility
One solution to this problem is to upgrade the system so that the actuators have greater maximum authority. However, this translates to an increase in size, weight, cost, and energy consumption that may not be acceptable for a given application. For example, it is undesirable to use a large manipulator on a mobile platform in a home application. The alternative is to use a control system in conjunction with an intelligent motion planner that effectively uses the dynamics of the system to generate feasible trajectories.

This paper is a generalization of the approach presented in [1] and describes experiments on a 2 degrees of freedom (DOF) manipulator. In contrast, experiments in [1] were only performed on a 1 DOF manipulator. Additionally, fast trajectory planning by learning from previous experience is introduced to significantly improve the planning time.

The research uses Sampling-Based Model Predictive Optimization (SBMPO) [2, 4, 5] as the foundation of an intelligent motion planner for input constrained robotic systems. SBMPO derives its name from its application to model predictive control (MPC). Like nonlinear MPC, SBMPO can be applied to systems with complex, nonlinear dynamics. However, unlike more standard MPC optimization, it does not rely on gradients or their derivatives. Instead, it discretizes the optimization problem by gridding the space of possible system inputs and uses an A*-type algorithm to help create and search the resulting graph. SBMPO relies on the use of an estimate of the cost-to-goal, i.e., the cost from the current state to the goal state. The closer the cost-to-goal estimate is to the actual cost, the faster SBMPO computes. Furthermore, if the cost-to-goal estimate is optimistic, such that it is a rigorous lower bound on the cost, the underlying A* algorithm is guaranteed to yield an optimal solution. It should be noted that in this research, SBMPO is not used with a receding horizon, but instead the optimization plans all the way to the goal.

A comparison between SBMPO and other nonlinear MPC approaches is presented in [6]. It is seen that unlike approaches based on gradients and Hessians, SBMPO has the capacity to avoid local minima and if no implicit grid is used in the output space, it is proved that SBMPO is guaranteed to find a global optimum subject to the sampling. Additionally, SBMPO has the capacity to dramatically increase its computational speed by learning cost-to-goal functions. This is the subject of ongoing research.

It should be mentioned that SBMPO has been applied to 4 DOF planning for the Barrett WAM robot arm [7]. The planning was based on the kinematic model of the manipulator and produced trajectories that reached the desired goals using relatively coarse input sampling (i.e., sample sizes ranging from 5 to 10). Since lifting for one arm uses at most 4 DOF, the prior results [7] indicate that the approach presented in this paper can be used to plan trajectories for heavy lifting using more complex manipulators.

A sampling based motion planner is introduced in [8]. However, unlike the randomized A* approach [2, 9], this approach is not based on optimization. Motion planners that do utilize optimization are discussed in [10], [11], but do not appear to be able to take advantage of the efficiency obtained by using the prediction associated with the optimistic estimate of cost-to-goal. SBMPO has previously been applied to input-constrained robot trajectory planning. It’s first application to the generation of steep hill trajectories for a robotic ground vehicle was presented in [2]. However, those initial results, which optimized distance travelled, required integration of the vehicle’s dynamic model, a computationally expensive task, and did not have a mechanism for specifying the desired velocity at the goal, which is critical for many practical applications. Subsequently, an SBMPO approach that optimizes a minimum time cost function was developed [1], [3]. Unlike the approach of [2], although used to enforce torque constraints, the dynamic model is not integrated and the goal velocity is specified. As SBMPO requires an integration model, the new approach was based on integrating an extended kinematic model, consisting of the standard kinematic model preceded by two integrators. This model is mathematically a dynamic model and enables SBMPO to sample the acceleration and provide the acceleration, velocity, and position as functions of time that are needed.

In [6], it is seen that unlike approaches based on gradients and Hessians, SBMPO has the capacity to avoid local minima and if no implicit grid is used in the output space, it is proved that SBMPO is guaranteed to find a global optimum subject to the sampling. Additionally, SBMPO has the capacity to dramatically increase its computational speed by learning cost-to-goal functions. This is the subject of ongoing research.

Although the results for heavy lifting of manipulators [1] and the related results of [3] are promising, they are only formulated and demonstrated on 1 degree of freedom (DOF) systems and the approach has no mechanism to learn from previously developed trajectories in order to speed up the computation of a new trajectory for a similar problem but with a different goal. The learning approach has some resemblance to a prior graph-based learning approach [12] that uses the entire graph (not just the optimal path) for a path planning problem as an initial condition to a new planning problem. Note that seeding a search [13], [14] and reusing previous computations in planning [15], [16] have already been demonstrated to work effectively. In contrast, the approach here considers trajectory planning, not simple path planning, and seeds the new trajectory planning problem with only the optimal trajectory from a prior trajectory planning problem. The results of this paper are developed for fully actuated, multiple DOF systems and the basic planning algorithm is experimentally verified for a heavy lifting problem involving a two-link (i.e., 2 DOF) manipulator.

To better understand the primary application considered in this paper, consider the 2-link manipulator shown in Fig. 1. When the manipulator load is light, the points at which the end effector can be held statically (i.e., the reachable region) are the kinematically achievable points, which constitute the connected region shown in Fig. 1(a). However, for sufficiently heavy loads, the torque limitations of the manipulator become important and the reachable region is a subset of the region.
in Fig. 1(a). In particular the reachable region becomes disjoint and consists of an upper region and a lower region as seen in Fig. 1(b). To move from the lower region to the upper region, trajectories with sufficient momentum must be developed, requiring the use of the system’s dynamic model.

It should be mentioned that previous research has studied minimum time control and trajectory generation for robotic manipulators [17], [18]. These studies consider the problem of determining manipulator control commands to move a manipulator in minimum time along a specified path subject to torque/force constraints. In contrast, the approach presented in this paper does not need a path to create a trajectory. In fact, given the initial and final positions and velocities, the trajectory is directly generated subject to torque/force constraints as a result of the planning process. However, sampling, which is used for fast computations, always leads to suboptimality; hence the trajectories developed by the methodology presented in this paper are never truly optimal, although some of the trajectories are nearly optimal.

The problem of heavy lifting for manipulators has been considered in prior research [19]. The previous approach has similarities and differences with the current approach. Both approaches are based on solving an optimization problem. However, here the cost function is simply time, whereas in [19] the cost function is non-physical and consists of a term that is a quadratic function of the manipulator joint trajectories. This latter term enables the approach of [19] to attempt to find the maximum load. In contrast, the current approach specifies the load and only searches for the optimal acceleration commands (and indirectly the optimal torques) for the given load, which is more traceable to how planning would be accomplished in practice. Both methods discretize the optimization problem. The approach of [19] accomplishes this by using B-Splines to represent the joint trajectories, whereas the current approach samples the commanded accelerations sent to a computed torque controller. The approach of [19] relies on Newton-type optimization, which requires an initial feasible trajectory and is subject to convergence to a local minimum depending upon the initial conditions. In contrast, the current approach optimizes using SBMPO, which does not require an initial trajectory and as discussed in [6] converges to the global optimum of the discretized problem. Finally, unlike [6], the current approach is explicitly seeking to develop a methodology that is computationally fast. Using 1-DOF of the PUMA 762 robot, Appendix A provides a comparison between the heavy lifting planning given in [19] and planning using the current approach. The objective was to lift a heavy load from vertically down to vertically up. It is seen that the current approach, which is based on solving a minimum time problem, requires 2.8 sec and 2 swings in contrast to the results of [19], which require 7.8 sec and 5 swings. Also, the torque profile of the current approach is very close to “bang-bang” whereas the torque profile of [19] does not exhibit the same bang-bang behavior.

In additional related work, an energy based control approach for a swing up control problem for underactuated system has been presented [20], [21] and can also be a solution to lifting heavy loads. However, unlike that work, the approach presented in this paper can work in the presence of obstacles [1] and can easily be extended to higher DOF systems.

This paper is organized as follows. Section 2 gives a brief description of the SBMPO algorithm underlying the proposed approach to trajectory planning and control. Section 3 uses SBMPO to plan trajectories using an extended kinematic model and presents the development of an optimistic estimate of the cost-to-goal that enables efficient generation of trajectories ending in zero velocity. Section 4 extends the results of Section 3 to take into account the actuator limitations and the dynamic model. Section 5 presents trajectory planning using a prior trajectory as a form of learning. Section 6 presents the experiments and results of the study. Lastly, Section 7 discusses conclusions and future work.

2 Sampling-Based Model Predictive Optimization

Sampling-Based Model Predictive Optimization (SBMPO) [2], [22] is a sampling-based, graph search algorithm for motion planning using models of a robotic system (more generally a dynamic system) and models of the environmental obstacles [23]. Fig. 2 shows the block diagram of a trajectory planning strategy that uses SBMPO, which integrates either the system’s kinematic or dynamic model and uses the obstacles and the dynamic model (when the kinematic model is integrated) to determine constraints that are enforced on the generated trajectory. The model, cost evaluation, and estimate of cost-to-goal are supplied by the user. It should be noted that in the SBMPO algorithm, a graph is created from start to goal and each vertex on the graph keeps track of the states of the system, the control input, and the cost associated with the state.

SBMPO can plan using a variety of cost functions, including the standard sum of the squared error cost function used in Model Predictive Control (MPC) [24]. SBMPO was motivated by a desire to employ sampling and A* -type optimization in place of the nonlinear programming that is commonly employed for optimization in MPC. This provides SBMPO with the ability to avoid local minima and have fast computations with a properly designed A* estimate of cost-to-goal.

2.1 SBMPO Algorithm

Algorithm 1 describes SBMPO, which has many of the features of the LPA* algorithm [25]. For each vertex v, the algorithm maintains h(v), g(v), and rhs(v), where h(v) represents the cost-to-go estimate (known in the artificial intelligence literature as the heuristic), g(v) is the start cost (i.e., the cost of a lowest cost trajectory from v_{start} to v), and rhs(v) is another
estimate of the start cost. For any vertex $v$, $status_v$, the status of the vertex is given by

$$status_v = \begin{cases} \text{consistent}, & g(v) = rhs(v) \\ \text{inconsistent}, & \text{otherwise.} \end{cases}$$  \hspace{1cm} (1)$$

where consistent vertices are vertices that have already been explored, while inconsistent vertices are vertices which need to be explored.

All inconsistent vertices are pushed into priority queue $Q$ in the order of their priority, which is a two-component key vector $\{2\}$. The keys are ordered lexicographically with the smaller key values having a higher priority. An implicit grid is
used by the algorithm to merge the vertices with similar states, thereby limiting the number of vertices that can exist within any finite region of the state space.

The main function Main() first calls Initialize() to initialize the trajectory-planning problem \{37\}. Initialize() sets the initial g-values of the vertices, \(v_{\text{start}}\) and \(v_{\text{goal}}\), to infinity and sets their rhs-values to 0 and \(\infty\) respectively. Thus, initially \(v_{\text{start}}\) is the only locally inconsistent vertex and is inserted into the otherwise empty priority queue \(Q\) with a key calculated according to \{1\}. The estimate of cost-to-goal value of \(v_{\text{goal}}\) is set to 0, and the estimate of cost-to-goal value of \(v_{\text{start}}\) is calculated with the help of \(\text{Heuristic\_Cost()}\) \{4\}.

SBMPO expands vertices by picking up the top vertex in \(Q\) \{39\} until \(v_{\text{goal}}\) is locally consistent or the key of the vertex to expand next is no less than the key of \(v_{\text{goal}}\) \{38\}. The top vertex is expanded by sampling the input space \{26\}, typically using Halton sampling \{26\} or random sampling. The number of samples is called the \textit{branchout factor}. The input sample (controls) and current state \textit{(state(u))} (i.e., state of the selected vertex \(u\)) are passed to the robot model, and the robot model is integrated to determine the new state \textit{(state(u'))} of the system \{28\}.

The new state \textit{(state(u'))}, if valid (i.e., it satisfies all constraints), is then added to the graph \{29\}. The graph is expanded with the help of \(\text{Create\_Vertex()}\) \{30\}, which uses an implicit grid \{27\} to check if the graph already contains a vertex with state close to \textit{(state(u'))}. If such a vertex exists, then return the vertex and add an edge from the current vertex (i.e., the selected vertex) to the vertex whose state is similar to the new state. Otherwise, return a new vertex \(u'\) whose state is the new state \textit{(state(u'))}.

The newly expanded vertex \(u'\) is added to the successor list of the selected vertex \(u\) \{32\}. The heuristic value \(h(u')\) of the new state is calculated by \(\text{Heuristic\_Cost()}\) \{4\}. SBMPO then updates the rhs-value and key-value of the new vertex as well as their membership in the priority queue if they become locally consistent or inconsistent with \(\text{Update\_States()}\) \{14\}, and finally calculates a trajectory by repeatedly expanding locally inconsistent vertices in the order of their priorities.

### 2.2 Modification of SBMPO Initialize Procedure

A modified SBMPO \textit{INITIALIZE} procedure is shown in Algorithm 2. Given a goal \(v_{\text{goal}}\), a resulting trajectory from the motion planner can be described as \(V_{r} = \{v_{1}, v_{2}, \ldots, v_{m}\}\), where \(v_{i}\) contains trajectory components of the joints 1 to \(n\), and \(m\) is the number of steps to reach the goal. Algorithm 2 uses information from \(V_{r}\), a baseline trajectory, for planning for a new goal \(v_{\text{new\_goal}}\). In contrast to planning \textit{a priori} information, where the priority queue only contains the start vertex \(v_{\text{start}}\), the priority queue is initialized with the vertices from the baseline trajectory \(V_{r}\).

Due to the change in the goal position (i.e., \(v_{\text{goal}}\) is replaced by \(v_{\text{new\_goal}}\)), new heuristics corresponding to the vertices \(V_{r}\) are computed using \(\text{Heuristic\_Cost()}\) \{7\}. The vertices \(V_{r}\) with updated \textit{key} values are pushed into \(Q\) according to their priority: \(Q = \{q_{1}^{\text{new}}, q_{2}^{\text{new}}, \ldots, q_{m}^{\text{new}}\}\), where \(q_{i}^{\text{new}}\) is the vertex with the highest priority (e.g. lowest cost) and will be expanded first \{8\}. The algorithm then expands vertices by picking up the top vertex in the pre-loaded \(Q\) until \(v_{\text{new\_goal}}\) is locally consistent or the key of the vertex to expand next is no less than the key of \(v_{\text{new\_goal}}\).

### 3 Trajectory Planning Using an Extended Kinematic Model

Although SBMPO can use the dynamic model as the integration model, as this model is typically complex, this leads to long computational times when computing optimal trajectories. On the other hand, kinematic models are relatively simple and computationally inexpensive to integrate. Hence, the approach here is to integrate the kinematic model and use the dynamic model to enforce constraints on the acceleration that help to ensure that the torque constraints are not violated.

The first question to answer is how do we use the kinematic model to develop the desired acceleration, velocity, and position vectors (i.e., \(\ddot{q}_{d}, \dot{q}_{d}, \text{ and } q_{d}\)) needed by the trajectory tracking controller? This can be accomplished letting the model in the trajectory planning approach described in Fig. 2 be the “extended kinematic model,” of Fig. 3, where \(q_{d} \in \mathbb{R}^{n}\), \(x \in \mathbb{R}^{m}\), \(m\) is the number of states, \(I_{n}\) is the \(n \times n\) identity matrix, and \(n\) denotes the number of DOF of the robotic system, which is assumed here to be fully actuated. In Fig. 3 and below \(T\) denotes the sample period for trajectory generation. It follows from the integration operations of Fig. 3 that

\[
\ddot{q}_{d}(NT) = \ddot{q}_{d}(NT - T) + T\dddot{q}_{d}(NT),
\]

(2)

\[
q_{d}(NT) = q_{d}(NT - T) + T\dot{q}_{d}(NT),
\]

(3)

where \(N\) is some positive integer.

Since \(\dddot{q}_{d}\) is the input to the extended kinematic model and SBMPO samples the input, SBMPO is able to generate \(\dddot{q}_{d}\) by sampling and \(\ddot{q}_{d}\), and \(q_{d}\) by model integration. Note that SBMPO must assume some bounds on \(\ddot{q}_{d}\), which are given by
Algorithm 1 SBMPO Algorithm

1: function KEY(s)
2:   return \([\min(g(s), rhs(s)) + h(s), \min(g(s), rhs(s))]\)
3: end function

4: function HEURISTIC Cost(s, v_goal)
5:   return minimum time to reach \(v_{goal}\)
6: end function

7: function INITIALIZE()
8:   OPEN \(\leftarrow\) \(\emptyset\) \(\triangleright\) OPEN is a priority queue
9:   \(g(v_{start}), g(v_{goal}), rhs(v_{goal}) \leftarrow \infty\)
10:  \(rhs(v_{start}), h(v_{goal}) \leftarrow 0\)
11:  \(h(v_{start}) \leftarrow HeuristicCost(v_{start}, v_{goal})\)
12:  OPEN \(\leftarrow\) OPEN \(\cup\) \(v_{start}\)
13: end function

14: function UPDATE STATE(v)
15:   if \(v \neq v_{start}\) then
16:     \(rhs(v) \leftarrow \min_{v' \in \text{Pred}(v)} (g(v') + \text{cost}(v', s))\)
17:   end if
18: end function

26: function GENERATE CHILDREN(u)
27:   for controls \(\in\) Samples(BRANCHOUT) do
28:     state(u') = RobotModel(state(u), controls)
29:     if IsValid(state(u')) then
30:       \(u' \leftarrow\) CreateVertex(state(u'))
31:       \(h(u') \leftarrow HeuristicCost(u', v_{goal})\)
32:       Succ(u) \(\leftarrow\) Succ(u) \(\cup\) u'
33:     end if
34:   end for
35: end function

36: function MAIN()
37:   Initialize()
38:   while OPEN.TopKey() < Key(v_{goal}) or rhs(v_{goal}) \(\neq\) g(v_{goal}) do
39:     \(u \leftarrow\) OPEN.pop() \(\triangleright\) Top element of the queue
40:     GenerateChildren(u)
41:     if g(u) > rhs(u) then
42:       g(u) \(\leftarrow\) rhs(u)
43:       for s \(\in\) Succ(u) do
44:         UPDATE STATE(s)
45:       end for
46:     else
47:       g(u) \(\leftarrow\) \(\infty\)
48:       for s \(\in\) Succ(u) \(\cup\) \{u\} do
49:         UPDATE STATE(s)
50:       end for
51:     end if
52:   end while
53: end function
54: return Trajectory
55: end function
Algorithm 2 Modified SBMPO INITIALIZE Function

1: function INITIALIZE()
2:   OPEN ← ∅
3:   g(v_{start}), g(v_{goal}), rhs(v_{goal}) ← ∞
4:   rhs(v_{start}), h(v_{goal}) ← 0
5:   OPEN ← OPEN ∪ v_{start} ⊢ prioritize using key(v_{start})
6:   for v ∈ V_{r} do
7:      h(v) ← HeuristicCost(v, v_{new}^{new}) ⊢ Update heuristic
8:      OPEN ← OPEN ∪ v ⊢ Insert the updated vertex into the Queue
9:   end for
10: end function

Fig. 3. Extended kinematic model [1]

\[
-a_i \leq \ddot{q}_i \leq b_i, \quad i = 1, \ldots, n, \quad (4)
\]

where \(a_i > 0\) and \(b_i > 0\). For a revolute \(n\) DOF manipulator these bounds are the bounds on the accelerations of the \(n\) joints. (There may also be bounds on the joint velocities and positions. SBMPO is used to enforce these constraints as discussed in Section 4.)

The estimate of cost-to-goal in Fig. 2 is optimistic if it is chosen to be zero and the termination criteria may be chosen such that SBMPO only stops when it reaches the goal at a desired velocity. However, in this case SBMPO becomes very computationally inefficient. As is well known, A\(^*\)-type algorithms become efficient when a fairly non-conservative, optimistic estimate of cost-to-goal is used. What is needed is an appropriate estimate of cost-to-goal corresponding to the minimum time cost.

Minimum Time Estimate of Cost-to-Goal

Motivated by (4), consider a 1 DOF system described by

\[
\ddot{q} = u; \quad q(0) = q_0, \quad \dot{q}(0) = \omega_0, \quad (5)
\]

where \(q\) and \(u\) are scalars and for \(a > 0\) and \(b > 0\), \(u\) is bounded by

\[
-a \leq u \leq b. \quad (6)
\]

The state space description of (5) is given by

\[
\dot{q}_1 = q_2; \quad \dot{q}_2 = u; \quad q_1(0) = q_0 \triangleq q_{1.0}, \quad q_2(0) = \omega_0 \triangleq q_{2.0}, \quad (7)
\]

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 \(q_{1,f} \triangleq q_f\). 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 we assume that \(q_{1,f} = 0\).

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 [28]. In fact, the above problem
Consider a dynamic model of a robot system given by

\[ M(q)\ddot{q} + C(\dot{q}, q) + G(q) = \tau, \quad (10) \]

where \( \ddot{q}, \dot{q}, \) and \( q \in \mathbb{R}^n \) are respectively the angular acceleration, velocity, and position, \( M \in \mathbb{R}^{nxn} \) is the inertia, \( C(\dot{q}, q) \in \mathbb{R}^n \) is the friction term, \( G(q) \in \mathbb{R}^n \) is the gravity term, and \( \tau \in \mathbb{R}^n \) is the applied torque.

Using a computed torque tracking controller [29], the inputs to the controller are the desired acceleration, velocity, and position (i.e., \( \dot{q}_d, \ddot{q}_d, \) and \( q_d \)) and the desired torque \( \tau_d \) is calculated as

\[
\tau_d = M(q)[\ddot{q}_d + K_v(\ddot{q}_d - \ddot{q}) + K_p(q_d - q)] \\
+ C(\dot{q}, q) + G(q),
\]

where \( M(q)\ddot{q}_d \) is the feedforward term, \( C(\ddot{q}, q) + G(q) \) are respectively the friction and gravity compensation terms, \( M(q)[K_v(\ddot{q}_d - \ddot{q}) + K_p(q_d - q)] \) is the feedback term, and \( K_v \in \mathbb{R}^{nxn} \) and \( K_p \in \mathbb{R}^{nxn} \) are the feedback gains.

The most direct method of taking into account the actuator limitations in planning is to simply choose the closed loop model as the SBMPO model [22]. The problem with this basic approach is that the controller is updated at a fast rate, e.g., 1 kHz, which must be the update rate for the closed-loop model. This leads to long planning times.
One alternative approach does not explicitly consider the tracking controller. Instead, it is assumed that the tracking controller is such that \( q(t) \approx q_d(t) \), \( \dot{q}(t) \approx \dot{q}_d(t) \), and results in the feedback terms in (11) being small compared to the remaining terms. In this case, the feedback torque computed by the tracking controller, given by (11), is dominated by the feedforward, friction and gravity compensation portions of the controller such that it is assumed that

\[
\tau_d(t) = M(q)\ddot{q}_d(t) + C(\dot{q}_d(t), q_d(t)) + G(q_d(t)) ,
\]

where in the experiments of Section 6.1, \( C(\ddot{q}_d, q_d) \) was set to zero. (Note that the \( q(t) \) and \( \dot{q}(t) \) in the \( C(\cdot) \) and \( G(\cdot) \) terms of (11) have been replaced by \( q_d(t) \) and \( \dot{q}_d(t) \) in (12).) Although (12) will sometimes underpredict the magnitude of \( \tau_d(t) \) and hence introduces the need for conservatism in the trajectory planning. This new approach uses the extended kinematic model, as in the previous section, but modifies the sampled (commanded acceleration) inputs to avoid saturation of the actuator torque.

Below we assume that \( t \in [NT, NT + T) \), where \( N \) is some positive integer. During this interval \( \ddot{q}_d(t) \) is held constant at its sampled value \( \ddot{q}_d(NT) \), i.e.,

\[
\ddot{q}_d(t) = \ddot{q}_d(NT), \ t \in [NT, NT + T).
\]

It follows that for \( t \in [NT, NT + T) \)

\[
\dot{q}_d(t) = \dot{q}_d(NT) + \ddot{q}_d(NT)t, \tag{14}
\]
\[
q_d(t) = q_d(NT) + q_d(NT)t + \frac{1}{2}\ddot{q}_d(NT)t^2. \tag{15}
\]

Ideally, one should check whether the torque \( \tau_d(t) \) given by (12) violated its saturation constraints \((-\tau_{\text{max},i} < \tau_{d,i}(t) < \tau_{\text{max},i}, \ i = 1, ..., n)\) in the entire interval \( t \in [NT, NT + T) \). However, to save computational time the torque will only be evaluated at the initial instant \( t = NT \), which introduces potential analysis error since the torque can violate the saturation constraints later in the interval. To account for this error and that associated with neglecting the feedback terms in (12), the assumed saturation range is reduced such that for some \( \bar{\tau}_i < \tau_{\text{max},i} \) it is determined whether

\[
-\bar{\tau}_i < \tau_{d,i}(NT) < \bar{\tau}_i, \ i = 1, ..., n. \tag{16}
\]

For each \( i \in \{1, ..., n\} \) if \( \tau_{d,i}(NT) > \bar{\tau}_i \), then \( \tau_{d,i}(NT) \leftarrow \bar{\tau}_i \) or if \( \tau_{d,i}(NT) < -\bar{\tau}_i \), then \( \tau_{d,i}(NT) \leftarrow -\bar{\tau}_i \). If torque saturation was detected in the above tests, then to choose a desired acceleration that does not violate the torque constraints, it follows from (12) that one can let

\[
\ddot{q}_d(NT) \leftarrow M(q)^{-1}[\tau_d(NT) - C(\dot{q}_d(NT), q_d(NT)) - G(q_d(NT))]. \tag{17}
\]

Notice that (17) ensures that at time \( NT \) the trajectory, which at that instant is given by \((\ddot{q}_d(NT), \dot{q}_d(NT), q_d(NT))\) has a torque requirement \( \tau_d(NT) \) that does not violate the torque constraints given by (16).

In general, there are also constraints on the joint velocities, which translate to bounds on the derivatives of the elements of \( q_d \), given by

\[
-c_i < \dot{q}_{d,i}(NT) < d_i, \ i = 1, ..., n. \tag{18}
\]

where \( c_i > 0 \) and \( d_i > 0 \). Obstacles can also lead to bounds on the joint positions. SBMPO enforces these output constraints by discarding all nodes corresponding to trajectory elements \((\ddot{q}_d(NT), \dot{q}_d(NT), q_d(NT))\) that violate the velocity or position constraints.
5 Fast Trajectory Planning By Learning from a Baseline Trajectory

Although the SBMPO procedure given above is designed to be efficient by avoiding integration of the dynamic model and using a non-conservative estimate of cost-to-goal based on the solution of a minimum-time optimal control problem, trajectory planning can be slower than desired (e.g., on the order of seconds for a 2-DOF manipulator) when the trajectory requires that one or more of the inputs reaches its upper bound, which in manipulator planning occurs when the load is sufficiently heavy. This appears to occur because the experimental method used to compute the bounds in (6) leads to an overly optimistic bounds for the corresponding degrees of freedom and hence the estimate of cost-to-goal (9) loses accuracy.

One way of speeding up computations for manipulator lifting is to precompute a series of baseline trajectories that correspond to various loads and “high lifts” of these loads. The resulting trajectory vertices can then be used to initialize the priority queue in SBMPO. This can substantially speed up the computations of SBMPO when the baseline trajectory is relevant to the current lifting problem, for example, when the current load is less than that of the baseline trajectory and the load is to be lifted to a position lower than that of the baseline trajectory. It will be seen in Section 6 that this procedure does lead to substantial computational speedup. The initialization of SBMPO is modified and the details are described in Algorithm 2, presented in Section 2.2.

A fundamental component of Algorithm 1 (SBMPO) is the priority queue Q, which contains all vertices that need to be explored. Whereas the standard SBMPO algorithm initializes Q with only the start and goal vertices, Algorithm 2 initializes Q with the estimate of cost-to-goals and vertices from a baseline trajectory. Since SBMPO propagates by expanding the top vertex (i.e., the one with lowest cost) in Q until the goal vertex is reached, Algorithm 2 accelerates the planning process by using information (when possible) from the baseline trajectory. In practice, this amounts to using the momentum characteristics of the first part of a baseline trajectory (i.e., some of the initial baseline vertices) to enable the manipulator to move from the lower reachable region to a higher region, possibly the upper reachable region depending upon the planning goals.

If the goal is identical to the baseline trajectory goal and the load is less than or equal to that of the baseline trajectory, SBMPO will simply reproduce the baseline trajectory since it has momentum characteristics that enable the load to reach the goal, i.e., it is a feasible trajectory. However, when the goal differs from that of the baseline trajectory, the latter part of the trajectory will deviate from that of the baseline trajectory. If the characteristics of the baseline trajectory are not useful to the current planning task, then SBMPO will give all or most of vertices of the baseline trajectory high costs and essentially plan as if the standard initialization were used.

6 Experiments and Results

This section discusses experiments involving a 2 DOF manipulator in which a computed torque controller was used to track SBMPO-generated trajectories, characterized by the acceleration, velocity, and position of each joint. An iCore 7 Duo 2.5 GHz processor was used for the computations and SBMPO was implemented using C++. SBMPO was used to plan a trajectory with a 50 Hz sampling rate, i.e., $T = 0.02$ sec for planning using the manipulator’s extended kinematic model and dynamic model. Fig. 5 shows the 2 DOF manipulator and its schematic. Table 1 shows the dynamic parameters, where $m_1$ is the mass of link 1, $l_1$ is the length of link 1, $l_{cm1}$ is the distance of the $cm_1$ (the center of mass of link 1) from joint 1, $I_1$ is the inertia of link 1, $m_2$ is the mass of link 2, $l_{cm2}$ is the distance of $cm_2$ (the center of mass of link 2) from joint 2, $l_2$ is the length of link 2, and $I_2$ is the inertia of link 2. The effect of the dynamic parameters of the load can be analyzed separately but to simplify the analysis, it is reflected to link 2. Hence, the dynamic parameters of link 2 vary with the load.
Table 1. Key Parameters of 2 DOF Manipulator System with 2.27 kg. (5lb) and 4.54 kg. (10lb) Load

<table>
<thead>
<tr>
<th>Parameter</th>
<th>No Load</th>
<th>Load 5lb</th>
<th>Load 10lb</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$ kg</td>
<td>2.883</td>
<td>2.883</td>
<td>2.883</td>
</tr>
<tr>
<td>$\ell_1$ m</td>
<td>0.375</td>
<td>0.375</td>
<td>0.375</td>
</tr>
<tr>
<td>$\ell_{cm_1}$ m</td>
<td>0.195</td>
<td>0.195</td>
<td>0.195</td>
</tr>
<tr>
<td>$I_1$ kg-m$^2$</td>
<td>0.034</td>
<td>0.034</td>
<td>0.034</td>
</tr>
<tr>
<td>$m_2$ kg</td>
<td>1.085</td>
<td>3.401</td>
<td>6.732</td>
</tr>
<tr>
<td>$\ell_2$ m</td>
<td>0.300</td>
<td>0.300</td>
<td>0.300</td>
</tr>
<tr>
<td>$\ell_{cm_2}$ m</td>
<td>0.220</td>
<td>0.274</td>
<td>0.285</td>
</tr>
<tr>
<td>$I_2$ kg-m$^2$</td>
<td>0.013</td>
<td>0.029</td>
<td>0.044</td>
</tr>
</tbody>
</table>

![Fig. 6. Quasi static reachable region of the 2DOF manipulator (a) no load (b) 2.27 kg (5 lb) load (c) 5.54 kg (10 lb) load](image)

The 2 DOF manipulator is driven by two 150W Maxon RE40 motors, each with a maximum continuous torque of 177 mNm. The motors are coupled with a Maxon GP52 gearing system that has a gear ratio of 66:1. Hence, the maximum continuous joint torques are given by $\tau_{max,1} = \tau_{max,2} = 66 \times 177$ mNm $= 11.7$ Nm. To introduce conservatism as discussed in the latter part of Section 4, the assumed saturation range is reduced to $\bar{\tau}_1 = \bar{\tau}_2 = 10$ Nm. The angular positions are sensed by encoders, with 500 pulse per motor revolution, which are directly attached to the motors. The motor drivers used in this study are configured in torque/current mode and have the ability to output current to the motors. The torque calculation, tracking controller implementation, angular position sensing, and velocity calculation from the joint position are implemented using a PIII 900 MHz computer running the QNX realtime operating system with a 1 kHz sampling rate.

Figs. 6(a), 6(b), and 6(c) show respectively the reachable regions for no load, a 2.27kg (5 lb) load, and a 5.54 kg (10 lb load). These regions were determined by considering all possible joint configurations and recording the configurations that require joint torques that are under the torque limits. As the load increases, the reachable region becomes disjoint due to torque saturation as seen in Fig. 6 (b) and (c). In these cases, for a task that requires movement from the lower region to a higher region, the trajectory must have sufficient momentum to traverse the unreachable regions. Although it is possible to have a kinematically feasible goal that is outside the reachable regions, the manipulator can reach the goal at a specified velocity but cannot hold its position. This case is relevant for throwing, which is not considered in this paper.

6.1 Trajectory Planning and Experimental Results Using Dynamic Model

Fig. 6 (a) shows the quasi-static reachable region when the manipulator is unloaded. A task is given to move from rest at (0 m, -0.675 m), where the end effector is vertically down, to rest at a goal position (0.0 m, 0.65 m). In (4) $n = 2$ and the
angular acceleration bounds were chosen to be $a_1 = a_2 = b_1 = b_2 = 10.0 \text{ rad/sec}^2$. These bounds were used to compute the minimum time estimate of cost-to-goal described in Section 3 and were determined using the conservative estimates of the maximum joint torques, i.e., $\bar{\tau}_1 = \bar{\tau}_2 = 10 \text{ Nm}$. These torques were applied to the manipulator without a load when the two joint angles were zero (i.e., the end effector was at its lowest possible position), but the joints were not allowed to pass 90 deg. to prevent the manipulator from being damaged. The maximum acceleration observed for joint one was 10.6 rad/sec$^2$ and for joint two was 12.72 rad/sec$^2$. Negative maximum torques were also applied and the joint acceleration magnitudes were almost identical. The bounds above, which were used to compute the heuristics in SBMPO, are smaller in magnitude than these maximum and minimum accelerations.

In (18) $n = 2$ and the velocity bounds were chosen to be $c_1 = c_2 = d_1 = d_2 = 10.0 \text{ rad/sec}$. These bounds were calculated as follows. The nominal actuator speed for the Maxon RE40 motors described above is 6490 rpm at the maximum nominal torque and hence at each joint the nominal speed is $6940/66 \text{ rpm} = 11.0 \text{ rad/sec}$. The value 10.0 rad/sec was used to add some conservatism to the velocity constraint.

An implicit grid (see the SBMPO Algorithm of Section 2.1) based on the two joint angular positions and two joint angular velocities was used. The grid discretized the joint angles at 0.1 rad and the joint velocities at 0.1 rad/s. The grid resolution is a design parameter used to keep computations small. If the grid resolution is made too fine, the priority queue can get very large and SBMPO will take a long time to converge. Seven Halton samples of the commanded acceleration (the input to the extended kinematic model) were used at each SBMPO iteration. This is the branchout factor of Section 2.1. A study of the effect of implicit grid resolution and branchout factors on the SBMPO solution is found in Section 4 of Chapter 4 of [30].

SBMPO computed a trajectory in 0.57 sec with a 50 Hz trajectory update rate (i.e., $T = 0.02 \text{ sec}$). Fig. 7 shows the resultant commanded accelerations, which are the inputs to the manipulator system. The suboptimality is seen by the deviation from bang-bang behavior and the non-smoothness of the commanded accelerations. Fig. 8 shows a snapshot of the path of the manipulator’s end effector and the unloaded reachable region. It is seen that the manipulator stays in the reachable region. Figs. 9 and 10 show the simulation result as the trajectory is fed to the tracking controller. Fig. 11 shows the experimental result and compares the desired and actual angular positions. The average tracking error for joint one is 0.032 rad and joint two is 0.021 rad.

In the next experiment, the manipulator has a load of a 2.27 kg (5 lb) and the dynamic parameters are shown in Table 1 under the 2.27 kg (5 lb) load. Referring to Fig. 5, the task is again to move the manipulator from rest at a starting position (0 m,-0.675 m) to rest at a goal position (0 m,0.65 m). SBMPO computed a trajectory in 22.8 sec. Fig. 12 shows the resultant commanded accelerations, which are the inputs to the manipulator system. As in the no loaded case, the suboptimality is seen by the deviation from bang-bang behavior and the non-smoothness of the commanded accelerations. Fig. 13 shows a snapshot of the path of the manipulator’s end effector and the 2.27 kg (5 lb) load reachable region. Figs. 14 and 15 show the simulation result as the trajectory is fed to the tracking controller. Fig. 16 shows the experimental result and compares the desired and actual angular positions. The average tracking error for joint one is 0.065 rad and joint two is 0.028 rad. Trajectory planning is next performed for the manipulator with a larger load of 4.54 kg (10 lb). The corresponding dynamic parameters are shown in Table 1. SBMPO computed a trajectory in 16.0 sec. and the result is shown in Fig. 17.

For 2.27 kg (5 lb) load, SBMPO trajectory reaches the desired position at zero velocity and is stabilized about this position by the trajectory tracking controller. The controller is able to track the trajectory despite the torque saturation due to the momentum associated with the trajectory. Notice from Figs. 13 and 17 that the manipulator moves in the opposite direction of the final swing to obtain the required momentum needed to pass the uncontrollable region as it swings to the
Fig. 8. End effector’s path of the 2 DOF manipulator with no load. The task was to move from rest at (0 m, -0.675 m), where the end effector is vertically down, to rest at a goal position (0.0 m, 0.65 m). The planning time was 0.57 sec.

Fig. 9. Simulation results: Trajectory planning and tracking results for the no load case. The desired and actual angular positions were virtually identical.

Fig. 10. Simulation results: Trajectory planning and tracking results for the no load case. The desired and actual angular velocities were virtually identical.

vertical position.
Fig. 11.  Experimental results: Trajectory planning and tracking results for the no load case. The desired and actual angular positions closely coincide. The average tracking error for joint one is 0.032 rad and joint two is 0.021 rad.

Fig. 12. For the 2.27 kg (5 lb) load SBMPO produced these commanded angular accelerations, which are the inputs to the manipulator system

Fig. 13. End effector’s path of the 2 DOF manipulator with 2.27 kg (5 lb) load. The task was to move from (0 m,-0.675m), where the end effector is vertically down, to rest at a goal position (0.0 m,0.65m). The planning time was 22.8 sec.

6.2 Fast Trajectory Planning Using a Baseline Trajectory
The results in Sec. 6.1 showed a short planning time of 0.57 sec for the no load case (see Fig. 8), the case in which the reachable region is connected. Fast planning time were seen consistently in other no load trajectory generation problems, so
Fig. 14. Simulation results: Trajectory planning and tracking results for 2.27 kg (5 lb) load. The desired and actual angular positions were virtually identical.

Fig. 15. Simulation results: Trajectory planning and tracking results for 2.27 kg (5 lb) load. The desired and actual angular velocities were virtually identical.

Fig. 16. Experimental results: Trajectory planning and tracking results for 2.27 kg (5 lb) load. The desired and actual angular positions closely coincide. The average tracking error for joint one is 0.065 rad and joint two is 0.028 rad.

this phenomena is fairly general. However, for trajectory planning with loads of 2.27 kg and 5.54 kg (see Figs. 13 and 17,
Fig. 17. End effector's path of the 2 DOF manipulator with 4.54 kg (10 lb) load. The task was to move from (0 m, -0.675 m), where the end effector is vertically down, to rest at a goal position (0.0 m, 0.65 m). The planning time was 16.0 sec.

respectively), cases in which the reachable region was disjoint, the planning times were on the order of 20 sec. To obtain fast planning times, the approach of Section 5 was used.

The baseline trajectory was chosen to be the trajectory of Section 6.1 corresponding to a 5.54 kg (10 lb) load moving from rest at (0 m, -0.675 m), where the end effector is vertically down, to rest at the goal (0 m, 0.65 m). SBMPO required 16.0 sec to generate this trajectory. When the approach of Section 5 was used, SBMPO regenerated this trajectory in 3 msec, an extremely fast, but expected result that was generated as a “sanity check.” Table 2 shows the computation times achieved with the same load but increasingly different goals. It should be noted that there was a significant reduction of computation times (1 to 3 orders of magnitude) achieved by using the baseline trajectory when the goal was in the same direction as the goal of the baseline trajectory, corresponding to a positive value of the x-coordinate of the goal. However, when the goals were in the opposite direction to the baseline trajectory goal, corresponding to a negative value of the x-coordinate of the goal, the computation time improvement was minor (at best a factor of about 3). Both sets of results were expected.

Table 3 shows the computation times achieved with a 2.27 kg (5 lb) load and changing goals. (Recall that the baseline trajectory assumed a 5.54 kg (10 lb) load, which is larger than the actual load.) Again, as long as the goal is in the same direction as the goal of the baseline trajectory, there was a significant reduction in computation times (3 orders of magnitude). When the goal was in the opposite direction, the computations times only improved by about a factor of 2.

The columns in Tables 2 and 3 labeled “with learning” show the efficacy of the learning approach of Section 5 in yielding fast computation times. The learning approach is one of the contributions of this paper. The columns in Tables 2 and 3 labeled “without learning” represent the results of [1] if used for a 2 DOF manipulator.

To use the fast trajectory planning method in practice, one can generate several baseline trajectories. The baseline trajectory with a load greater than, but otherwise closest to, the actual load and a goal that is closest to the desired goal would then be used when generating a new trajectory.

7 Conclusions

The fundamental contribution of this paper was the development of a trajectory planning methodology for input constrained robotic systems that focuses on fast computations. The primary algorithm used was Sampling Based Model Predictive Optimization (SBMPO), a graph search algorithm that pseudo-randomly or randomly samples the inputs to a dynamic system and uses a version of A* optimization to find the optimal trajectory on the graph. Although SBMPO can directly integrate a complex dynamic model of a system, to enable fast computations, here SBMPO integrates an “extended kinematic model,” i.e., the kinematic model preceded by two integrators. This enables the development of the accelerations and velocities needed to define a trajectory that is followed using a computed torque controller.

Another key to fast computations is the development of a relatively non-conservative A* estimate of cost-to-goal. As minimum-time trajectories are desired in this work, an estimate cost-to-goal is developed based on the solution to a simple minimum-time optimization problem involving a double integrator. The solution involves only the solution of scalar quadratic equations. However, as the acceleration bounds associated with this estimate of cost-to-goal are developed for the no-load case, they become overly conservative for some of the loaded cases, leading to unacceptably large computation times (10 sec to 40 sec for the two link manipulator used in this study). To reduce these computation times, the process of generating a new trajectory with SBMPO can be seeded with a former trajectory, e.g., generated off-line. For the two link manipulator, the results show that this learning process can lead to reductions in computations of 1 to 3 orders of magnitude.
Table 2. Planning time comparison using a learned trajectory for 10 lb load. The experiments are done with 10 lb load.

<table>
<thead>
<tr>
<th>Goal ([m],[m])</th>
<th>without learning [s]</th>
<th>with learning [s]</th>
<th>nodes reused</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.00, 0.65)</td>
<td>16.031</td>
<td>0.003</td>
<td>110</td>
</tr>
<tr>
<td>(0.10, 0.60)</td>
<td>21.942</td>
<td>0.038</td>
<td>92</td>
</tr>
<tr>
<td>(0.20, 0.60)</td>
<td>12.232</td>
<td>0.900</td>
<td>82</td>
</tr>
<tr>
<td>(0.30, 0.60)</td>
<td>13.703</td>
<td>0.011</td>
<td>90</td>
</tr>
<tr>
<td>(0.30, 0.50)</td>
<td>11.301</td>
<td>0.524</td>
<td>57</td>
</tr>
<tr>
<td>(0.25, 0.50)</td>
<td>11.387</td>
<td>0.854</td>
<td>64</td>
</tr>
<tr>
<td>(0.60, 0.20)</td>
<td>14.553</td>
<td>0.891</td>
<td>61</td>
</tr>
<tr>
<td>(0.60, 0.30)</td>
<td>21.126</td>
<td>0.232</td>
<td>67</td>
</tr>
<tr>
<td>(-0.1, 0.60)</td>
<td>16.215</td>
<td>6.570</td>
<td>67</td>
</tr>
<tr>
<td>(-0.2, 0.60)</td>
<td>16.564</td>
<td>5.480</td>
<td>67</td>
</tr>
<tr>
<td>(-0.3, 0.60)</td>
<td>11.301</td>
<td>5.069</td>
<td>67</td>
</tr>
<tr>
<td>(-0.3, 0.50)</td>
<td>37.599</td>
<td>10.040</td>
<td>109</td>
</tr>
<tr>
<td>(-0.4, 0.50)</td>
<td>15.170</td>
<td>8.760</td>
<td>68</td>
</tr>
</tbody>
</table>

Table 3. Planning time comparison using a learned trajectory for 10lb load. The experiments are done with 5 lb load.

<table>
<thead>
<tr>
<th>Goal ([m],[m])</th>
<th>without learning [s]</th>
<th>with learning [s]</th>
<th>nodes reused</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.00, 0.65)</td>
<td>22.772</td>
<td>0.018</td>
<td>110</td>
</tr>
<tr>
<td>(0.10, 0.60)</td>
<td>18.683</td>
<td>0.017</td>
<td>89</td>
</tr>
<tr>
<td>(0.30, 0.60)</td>
<td>34.771</td>
<td>0.009</td>
<td>89</td>
</tr>
<tr>
<td>(0.30, 0.50)</td>
<td>14.723</td>
<td>0.019</td>
<td>79</td>
</tr>
<tr>
<td>(0.25, 0.50)</td>
<td>9.877</td>
<td>0.011</td>
<td>79</td>
</tr>
<tr>
<td>(-0.1, 0.60)</td>
<td>11.223</td>
<td>8.180</td>
<td>108</td>
</tr>
<tr>
<td>(-0.2, 0.60)</td>
<td>16.567</td>
<td>8.550</td>
<td>108</td>
</tr>
<tr>
<td>(-0.3, 0.60)</td>
<td>43.789</td>
<td>8.994</td>
<td>109</td>
</tr>
<tr>
<td>(-0.3, 0.50)</td>
<td>11.796</td>
<td>9.775</td>
<td>109</td>
</tr>
<tr>
<td>(-0.4, 0.50)</td>
<td>38.581</td>
<td>9.787</td>
<td>109</td>
</tr>
</tbody>
</table>

The fundamental results were experimentally demonstrated using a two link manipulator designed for these experiments. The experiments validated the methodology and showed that the computed torque controller was able to follow the trajectories generated by SBMPO.

Ongoing work is focused on the implementation of the proposed approach on higher degree of freedom manipulators and autonomous space vehicles. Additionally, trajectory planning applied to throwing a heavy object is being studied.

Appendix A: Result Comparison Between [19] and SBMPO

Figures 18, 19, and 20 show the planning results given in [19] and those produced using Sampling Based Model Predictive Optimization (SBMPO), presented in this paper, for 1 degrees of freedom (DOF) of a PUMA 762 robot lifting a 29.4 kg load using only joint 4 from 0° (vertically down) to 3.14 rad (vertically up); the load is attached 0.25 m from the joint. To be consistent with [19], the joint torque is limited to 0.75τ\textsubscript{max}, where from the PUMA 762 robot manual [31] τ\textsubscript{max} is 70 Nm. Also, the maximum angular velocity is set to 0.8\dot{q}\textsubscript{max}, where from [31] \dot{q}\textsubscript{max} is 2.62 rad/sec such that for SBMPO, in
Fig. 18. Angular position: planning results of [19] and SBMPO for 1 DOF of the PUMA 762 robot

Fig. 19. Angular velocity: planning results of [19] and SBMPO for 1 DOF of the PUMA 762 robot

(18), \( n = 1 \) and \( c_1 = d_1 = 2.62 \text{ rad/sec} \).

For SBMPO, in (4) \( n = 1 \) and the angular acceleration bounds were chosen to be \( a_1 = b_1 = 7.5 \text{ rad/sec}^2 \), where the acceleration bound was obtained from [31]. An implicit grid (see the SBMPO Algorithm of Section 2.1) based on the joint angular position and joint angular velocity was used. The grid discretized the joint angle at 0.05 rad and the joint velocities at 0.05 rad/s.

Figures 18 through 20 and 19 show that the trajectory of [19] required 7.8 sec for the load to reach its final position at zero velocity and consisted of 5 swings. In contrast, the SBMPO trajectory required only 2.8 sec and 2 swings. This is not a surprising result, as SBMPO solves a minimum time problem whereas [19] minimizes time along with other quantities. Correspondingly, the SBMPO torque profile of Fig. 20 is “bang-bang” whereas the torque profile corresponding to [19] does not exhibit the same bang-bang behavior. Note that the torque results of [19] are given in a scaled form, but here in their unscaled form. It was verified using the 1 link dynamic model that the position profile of Fig. 18 and the velocity profile of Fig. 19 do yield the unscaled torque given in Fig. 20.

References


Fig. 20. Joint torque: planning results of [19] and SBMPO for 1 DOF of the PUMA 762 robot