Energy optimization of trajectories for high level scheduling

Energy optimization of trajectories for high level scheduling

Oskar Wigström and Bengt Lennartson

***Abstract*— Minimization of energy consumption is today an issue of utmost importance in manufacturing industry. A previously presented technique for scheduling of robot cells, which exploits variable execution time for the individual robot operations, has shown promising results in energy minimization. In order to slow down a manipulator's movement the method utilizes a linear time scaling of the time optimal trajectory. This paper attempts to improve the scheduling method by generating energy optimal data using dynamic time scaling. Dynamic programming can be applied to an existing trajectory and generate a new energy optimal trajectory that follows the same path but with another execution time. With the new method, it is possible to solve the optimization problem for a range of execution times in one run. A simple two-joint planar example is presented in which energy optimal dynamic time scaling is compared to linear time scaling. The results show a small decrease in energy usage for minor scaling, but a significant reduction for longer execution times.**

I. Introduction

When designing the next generation of robotic manufacturing systems one must consider the ever growing demands on environmental as well as economical sustainability. Reducing the amount of expended energy during production contributes to both of these goals. In [12], a scheduling algorithm for decreasing energy usage in production systems was presented. The algorithm utilizes the fact that individual robot movements do not necessarily have to be performed in a time optimal fashion.

Even though minimal cycle time is often preferable, this does not imply that every action has to be performed at maximum speed in order to uphold an optimal cycle time. It might also be the case that a single production cell constrains the cycle time of other cells in series or parallel. If this is the case, it might not be necessary to run the other constrained cells based on time optimal schedules. One of the building blocks in the optimization model in [12] is the energy consumption $E_i(t_{fi})$ for each individual robot operation i as a function of its execution time t_{fi}. These local energy functions are then used to determine the globally optimal execution times t^*_{fi} for all operations, such that the overall energy of the system is minimized.

To prevent confusion, the terms path and trajectory are defined as follows. A path is a purely geometrical description of movement, while a trajectory is a path as a function of time. This means that velocity and acceleration can be defined for a trajectory at each coordinate along its path. The energy functions in [12] are based on a uniform time scaling of the time optimal robot trajectory, i.e a stretching of the acceleration profile. The resulting functions are most often convex and have the beneficial property that no torque constraints are violated. However, a linear time scaling of a trajectory does not equate to an energy optimal trajectory, and the algorithm may be improved by finding the energy optimal solution. The scheduling problem is modeled as a mixed integer linearly constrained problem, with a nonlinear cost function expressing the energy consumption of the cell. Generation of the energy consumption functions for the individual operations is a preprocessing step. Thus adding new local optimal energy functions requires no significant changes to the scheduling algorithm.

The focus of this paper is that of generating local optimal energy functions $E^*_i(t_{fi})$. This is done for each individual operation of the moving devices in a system, for instance a robot cell. Deriving $E^*_i(t_{fi})$ is synonymous with solving the minimum trajectory planning problem for a range of t_{fi}, with the geometric path of operation i as an input.

A trajectory planning problem can be described as generating the set of control inputs that will move a manipulator along a predefined geometric path without violating any dynamic or kinematic constraints. Usually trajectory planning problems are concerned with the optimization of some cost function, most often comprised of time, torque, jerk or a weighted combination of these.

Trajectory planning has been an area of research since the early 1970s [4], but at that time neither variable torque- nor path constraints were considered. Path constraints are a necessity for collision avoidance, and not until the mid 1980s [10], [1] the problem formulation was extended including this property. These early works were mainly focused on time optimal path planning. An excellent overview of the last three decades can be found in [3]. In summary, after minimum time, focus shifted towards minimum energy, which produces smoother trajectories and smaller tracking errors.

There is a large number of approaches to the minimum energy problem. Early attempts used dynamic programming [11], and later iterative dynamic programming [2], parameterized b-splines [6], Pontryagins maximum principle [9], among others, were used. Some of the later methods allow for constraints on the jerk and results in a continuous acceleration. Minimizing and constraining jerk is also a topic of interest as it reduces stress on the robot structure and gives better tracking.

This paper uses dynamic programming in order to solve the trajectory planning problem for a range execution times simultaneously. The grid required is as small as two dimensions and yields an optimal trajectory with discontinuous acceleration. If the grid is extended to three dimensions as

O. Wigström, B. Lennartson, Automation Research Group, Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden, oskar.wigstrom@chalmers.se

suggested in [11], continuous acceleration and a bounded jerk could be achieved. However, this would lead to a significantly increased computational effort. If the two dimensional grid is of high enough resolution and the size of the acceleration discontinuities are constrained, taking jerk into further consideration might not be necessary.

In contrast to [11], which employs a weighting between cost and time, our method includes elapsed time in the optimization model. This implies that, while [11] is based on free final time in the optimization, our model can generate solutions for specific final times. Note that, dynamic programming, as opposed to other methods, puts no bounds on model complexity or objective function for this problem. The main advantage of dynamic programming in this context however, is that it yields optimal solutions for the entire grid. In our method, the optimization model is formulated in such a way that all execution (final) times are included in the grid, and thus the dynamic programming optimization only needs to be run once. Even though the complexity of [11] is somewhat lower than ours, the dynamic programming optimization will have to be run once for every point in the energy function, $E_i^*(t_{fi})$. This means that it is a viable method if only a low resolution energy function with no requirements on specific final times is sought, i.e only a limited number of weightings are evaluated. Also note that since the optimization parameter is a single time varying variable, the dimensionality will be unchanged for additional robot joints as well as more intricate energy models.

The rest of this paper is structured as follows. Section II contains background on the minimum energy trajectory planning problem. Section III covers the optimization model and the dynamic programming algorithm structure. Section IV provides a numerical example with a comparison to linear time scaling. Finally, in Section V conclusions are drawn along with a brief discussion.

II. Problem Formulation

As previously established, for each operation, a trajectory planning problem needs to be solved for a range of execution times. Since this method is applied to one operation at a time, the operation index i is now discarded. Solving a trajectory planning problem entails finding the input torques required to move a manipulator along a predefined geometric path, while upholding its dynamical constraints. The joint torques of the manipulator can be expressed by a Lagrange formulation, see [8], pp. 131-140. Einstein summation convention is used, as in [11], where an index appearing in both the subscript and the superscript of a term constitutes a summation over its elements. The torque, T_i acting on the i :th joint can be expressed as

$$T_i = J_{ij}\ddot{q}^j + C_{ijk}\dot{q}^j\dot{q}^k + F_{ij}\dot{q}^j + G_i \tag{1}$$

where J_{ij} is the intertia matrix, C_{ijk} the tensor of centrifugal and Coriolis coefficients, F_{ij} the viscous friction matrix, G_i the gravitational vector and q^i the angular position of joint i. Also, $\dot{q}^i$ and $\ddot{q}^i$ represent the 1:st and 2:nd time derivative of q^i.

Let the geometric path be defined by a function $q_0(\tau)$, a parameterized curve dependent on one single variable $\tau(t)$. In this paper, the time optimal trajectory is used to define q_0. This implies that τ is the time scale for the time optimal trajectory, q_0. The relationship between q and q_0 can therefore be expressed as

$$q(t) = q_0(\tau(t)), \qquad 0 \leq \tau \leq \tau_f, \tag{2}$$

where $\tau(t)$ is an monotonically increasing function with a starting value of 0 and final value τ_f. The derivatives of q_0 with respect to τ is the same as those of the time optimal trajectory with respect to time. Differentiating (2) with regard to time yields expressions for speed and acceleration which are needed for computing the cost function and upholding constraints,

$$\dot{q}^i(t) = \frac{dq_0^i(\tau)}{d\tau}\dot{\tau} \tag{3}$$

$$\ddot{q}^i(t) = \frac{dq_0^i(\tau)}{d\tau}\ddot{\tau} + \frac{d^2q_0^i(\tau)}{d\tau^2}\dot{\tau}^2 \tag{4}$$

Further, combining (3) and (4) with (1) results in an expression for the torque as a function of τ and q_0,

$$\begin{aligned} T_i = {} & J_{ij}\frac{dq_0^j(\tau)}{d\tau}\ddot{\tau} + J_{ij}\frac{d^2q_0^j(\tau)}{d\tau^2}\dot{\tau}^2 + \\ & + C_{ijk}\frac{dq_0^j(\tau)}{d\tau}\frac{dq_0^k(\tau)}{d\tau}\dot{\tau}^2 + \\ & + F_{ij}\frac{dq_0^j(\tau)}{d\tau}\dot{\tau} + G_i \end{aligned} \tag{5}$$

The optimization procedure is also subject to a number of dynamic constraints, such as limits on torque, acceleration and speed.

An arbitrary cost function can be used, but in this paper it is of interest to examine minimum energy trajectories for specific execution times t_f. A simple energy model where the torque is assumed to be proportional to the current is adapted. This formulation suggests that the squared torque is proportional to the power. The cost as a function of t_f can with these assumptions be expressed as the sum of the squared joint torques

$$E(t_f) = \int_0^{t_f} T_iT^i \; dt = \int_0^{t_f} \left(\sum_{i=1}^{n} T_i^2\right) dt \tag{6}$$

Here, n is the number of joints. With the torque, T_i defined as in (5), the cost E is a functional of q_0 and τ. Since the former is known, solving the optimization problem is a matter finding τ while minimizing the cost and upholding dynamic constraints. The optimal cost $E^*(t_f)$ is the local energy function sought for each operation.

III. Dynamic Programming Method

Dynamic programming is an optimization method which can be applied to problems where a series of decisions need to be made and the dynamics of the system can be determined from any location within a state space. The objective is to determine how the state space can be traversed

in order to minimize a specified cost function. A discrete gridding of the state space generally has to be performed, and the choice of resolution is governed by requirements on the quality of the solution as well as limitations on computational power. Each decision in the decision process represents a movement within this grid, either from point to point or, where this is not possible, from any location where a grid point is reachable. For a detailed account of the theory behind the dynamic programming algorithm applied to discrete optimal control problems, see for example [5] or [7].

A. Modeling

Introduce t_k as a time instance and h_k as a variable sampling time, associated with the time index k such that

$$t_{k+1} \quad = \quad t_k + h_k \tag{7}$$

Note that $t_0 = 0$. This means that the time optimal time scale τ is updated for each time step (increment of k) as

$$\tau(t_{k+1}) = \tau(t_k + h_k) = \tau(t_k) + \Delta_k \tag{8}$$

where Δ_k can be regarded as a user defined sampling period or gridding of τ. Using this definition of τ implies that (2), at the sampling instance, can be described as

$$q_0(\tau(t_k)) \quad = \quad q(t_k) \tag{9}$$

Also, τ is mapped onto t such that both q and q_0 share the same path, although not necessarily the same velocity and acceleration. Equations (7)-(8) are illustrated in Fig. 1, where we emphasize that h_k is determined by the optimization while Δ_k is defined by the user. Define the second time derivative of τ as

$$\ddot{\tau}(t) = u(t_k), \qquad t_k \le t \le t_{k+1}, \tag{10}$$

where $u(t_k)$ is a piecewise constant control input. The decision to use a constant $\ddot{\tau}$ is an abstraction that will restrict the dimensionality of the problem to two. Even though it will introduce small discontinuities in the acceleration through (3), these minor artifacts can be considered marginal. One could instead choose to define the 3:rd derivative of τ as constant, and instead achive only discontinous jerk, but at the cost of dimensionality and complexity.

Integration of equation (10) yields the following difference equations

$$\dot{\tau}(t_k + h_k) \;=\; u(t_k)\; h_k + \dot{\tau}(t_k) \tag{11}$$

$$\tau(t_k + h_k) \;=\; \frac{1}{2}\; u(t_k)\; h_k^2 + \;\dot{\tau}(t_k)\; h_k + \tau(t_k) \tag{12}$$

With these equations, τ can be expressed using t_k and $\dot{\tau}(t_0)$. The optimal trajectory and cost can thus be described using only these variables. To solve the optimal control problem, a grid of possible values of t_k and $\dot{\tau}(t_k)$ is also necessary. If the optimal cost for a location in this grid for a specific k is denoted $J_k^*(t_k, \tau(t_k))$, then the functional

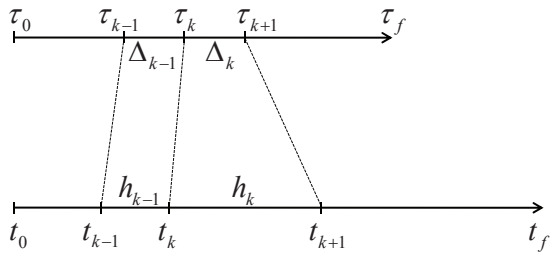

Fig. 1. Uniformly spaced τ mapped onto t.

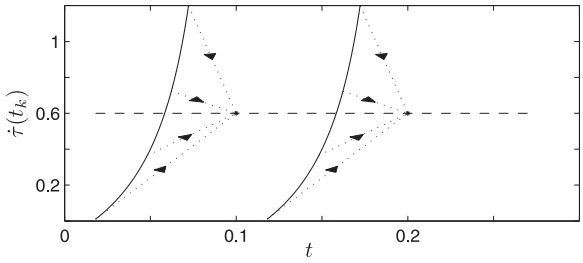

Fig. 2. The solid line denotes $(t_k, \dot{\tau}(t_k))$ and the two points show $(t_{k+1}, \dot{\tau}(t_{k+1}))$. The dashed line represents a constant $\dot{\tau}(t_{k+1}) = 0.6$. The dotted lines illustrates that the point $(t_{k+1}, \dot{\tau}(t_{k+1}))$ is reachable from $(t_k, \dot{\tau}(t_k))$.

equation of dynamic programming in the forward direction can be written as

$$\begin{aligned} J_{k+1}^*(t_{k+1}, \dot{\tau}(t_{k+1})) \quad &= \quad \min_{u(t_k)} [V(t_k, \dot{\tau}(t_k), u(t_k)) + \\ &\quad + \; J_k^*(t_k, \dot{\tau}(t_k))] \end{aligned} \tag{13}$$

Here, V is the cost of moving from $(t_k, \dot{\tau}(t_k))$ to $(t_{k+1}, \dot{\tau}(t_{k+1}))$, i.e the cost in (6), but with the integration from t_k to t_{k+1}. Then, for each $(t_{k+1}, \dot{\tau}(t_{k+1}))$ point in the grid, compute J_{k+1}^*. With equations (7)-(13), there is only one degree of freedom in J_{k+1}^* which is $u(t_k)$. Even though the possible solutions may not coincide in specific $(t_k, \dot{\tau}(t_k))$ grid points, it is still possible to interpolate values from J_k^*. Solving (13) for all k will result in the optimal cost for all the points in the grid.

Specifying the trajectory using h_k is effectively the same as using t_k. With (8) we can reformulate (11) and (12) to express h_k as

$$h_k \quad = \quad 2\; \frac{\Delta_k}{\dot{\tau}(t_{k+1}) + \dot{\tau}(t_k)} \tag{14}$$

In Fig. 2, note that the shape of the $(t_k, \dot{\tau}(t_k))$ curve does not change for t_k along a constant $\dot{\tau}(t_{k+1})$ since (14) does not depend on the explicit starting time t_k. This implies that the cost function does not have to be evaluated separately for each t_k where the value of $\dot{\tau}(t_{k+1})$ is the same. If instead $\tau/\dot{\tau}$ had been used for a grid, all iterations would be time invariant and all cost function evaluations could be performed at the first iteration and reused later. However, if a nonuniform gridding of t is used to significantly speed up computations, cost function evaluations from previous iterations cannot be reused. Our formulation will thus allow for a significantly decreased computational burden. Most

time is spent on interpolations of J_k^*, as this needs to be computed seperatly for each grid value of t_k.

It is also necessary to compute $u(t_k)$ to evaluate the cost function. Some further algebraic manipulation gives

$$u(t_k) \quad = \quad -2\,\frac{\Delta_k}{h_k^2} + 2\,\frac{\dot{\tau}(t_{k+1})}{h_k} \tag{15}$$

The expressions (14) and (15) are now used to compute steps in the grid, as well as evaluating the cost function. At the final iteration N, the resulting matrix J_N^* will hold the minimum cost for every grid point. Since t_k represents elapsed time, $t_N = t_f$, and as such the minimum cost for all execution times within the range of t_N can be found in this final J_N^*.

B. Algorithm

To start off, the first and second derivative of q_0 with respect to τ in (3) and (4) will have to be computed. As previously mentioned, the velocity and acceleration of the time optimal trajectory can be used to define these. Since $\ddot{\tau}$ is defined as constant between the discrete time updates, this leads to every point $(t_{k+1}, \dot{\tau}(t_{k+1}))$ in the grid having a corresponding $(t_k, \dot{\tau}(t_k))$ curve from where that point can be accessed at step $k+1$. The minimization part in (13) consists of sampling along this curve and choosing the point where the combination of J_k^* and V_k yields the lowest cost. Since the sampling of J_k^* is not available from a specific grid point, be reminded that the cost will have to be interpolated.

As for the algorithm, define a structure *opt* that can be used to store settings for the optimization problem. This includes grid size, grid resolution, limits on parameters, path parameters, constraints on dynamics and various sampling resolutions etc. Note that matrix multiplication is used at lines 11, 12 and 14 in the routine OPTIMALSOURCE to clone vectors into matrices. A Matlab like notation is used where for example $A(:,1)$ corresponds to all elements in the first column of a matrix A and $A(1:2,1)$ denotes the first 2 row elements in the first column.

```
DP-ALGORITHM(opt)
1  J ← NaN
2  J(1:2,:,1) ← 0
3  for k ← 1 to N_τ
4    do for e ← 1 to N_dτ
5         do J(:,e,k+1) ←
6              OPTIMALSOURCE(k, e, J(:,:,k), opt)
7  return J
```

In DP-ALGORITHM , the three-dimensional array J is used to store the optimal cost to reach a point in the grid for every time step. Its first two indices correspond to the values of the grid (transformed to integers) and the third the time step. Observe that since the sampling period h_k is varying the time step k is not generally the same as the integer transformation of the grid value t_k. Mark initial states with zeros. For each time step k, iterate over each value e in the $\dot{\tau}$ grid and call OPTIMALSOURCE. This function will compute the optimal value for all t in the grid along the specified $\dot{\tau}$ for time step k. When execution ends and J has been computed, the optimal cost for each time instance along the t-axis of the grid can be found by retrieving the smallest element in $J(t,:,\text{end})$. To find the optimal trajectory for a specific execution time, J can be used to trace the optimal path through the grid.

```
OPTIMALSOURCE(k, e, J(:,:,k), opt)
1  τ̇_k = opt.τ̇_min : opt.τ̇_res : opt.τ̇_max
2  h_k ← EQUATION14(τ̇_k, τ_{k+1}, τ_k, τ̇_{k+1})
3  u_k ← EQUATION15(τ̇_k, τ_{k+1}, τ_k, h_k)
4  j ← FINDOUTOFBOUNDELEMENTS(h_k, u_k, opt)
5  if 0 < length(j)
6     then τ̇_k(j) ← remove
7          h_k(j) ← remove
8          u_k(j) ← remove
9  c1 ← COSTFUNCTION(τ̇_k, h_k, u_k, k, opt)
10 t_{k+1} ← GETTIMEINDICES(J(:,:,k), opt)
11 T_k ← t_{k+1} * ones(1,:) − (h_k * ones(1,:))^T
12 Ṫau_k ← (τ̇_k * ones(1,:))^T
13 C2 ← INTERPOLATE(J(:,:,k), Ṫau_k, T_k, opt)
14 C1 ← (c1 * ones(1,:))^T
15 C ← C1 + C2
16 f ← MIN(C, row)
17 J_{e,k+1} ← FILLUP(f, t_k, opt)
18 return J_{e,k+1}
```

The first 3 rows can be summarized by sampling $\dot{\tau}$ and using (14) and (15) to calculate the possible sampling periods h_k, i.e compute the shape of the solid curve in Fig. 2. If this operation generates any h_k or u_k that are outside specified limits, these are purged (rows 4-9). Next, the cost function is evaluated. This can be implemented for example by computing a number of samples from the trajectories of the suggested parameters. Now compute which time indices in the grid that are relevant, i.e discard values that cannot be reached. On rows $11-12$, two matrices are generated that contain the t_k and $\dot{\tau}(t_k)$ values from where each t_{k+1} is reachable. Interpolate the values from $J(:,:,k)$ based on these coordinates and store in $C2$. Clone the values in $c1$ and store in $C1$ so that there is one copy of $c1$ for each row in $C2$. Sum $C1$ and $C2$ into C and use f to store the minimum element of the rows in C, these are the optimal costs for t_{k+1}. All that needs to be done now is to add NaN elements into f to correct for the time indices that were discarded on row 10, this result is then returned in $J_{e,k+1}$.

IV. A COMPUTATIONAL EXAMPLE

To demonstrate the presented method, an example with a frictionless two-link planar robot arm has been supplied. Table I provides the specifications for the example. The torque requirement for the motors at any instance can be expressed by the following equations. Their derivation can be found in [8], pp. 148-151.

TABLE I

EXAMPLE PARAMETERS

Const.	Description	Value
m_{li}	Mass of the i:th link	50 [kg]
m_{mi}	Mass of the i:th join motor rotor	5 [kg]
l_i	Distance, i:th link center of mass from axis	0.5 [m]
a_i	Length of the i:th link	1 [m]
I_{li}	Moment of inertia for the i:th link	10 [kg m^2]
I_{mi}	Moment of inertia for i:th join motor rotor	0.01 [kg m^2]
k_{ri}	Gear reduction ratio of the i:th motor	100
g	Standard gravity	9.807[m/s^2]

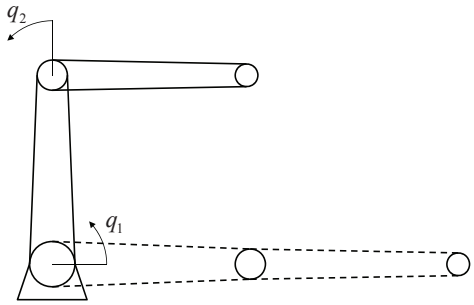

Fig. 3. In this example, the robot arm moves downwards from its upwards position. Note that the arrows represent the positive direction of q_1 and q_2.

$$u_1 = (I_{l1} + m_{l1}l_1^2 + k_{r1}^2 I_{m1} + I_{l2} + m_{l2}(a_1^2 + l_2^2 + \\ +2a_1 l_2 c_2) + I_{m2} + m_{m2}a_1^2)\ddot{q}_1 + (I_{l2} + m_{l2}(l_2^2 + \\ +a_1 l_2 c_2) + k_{r2}I_{m2})\ddot{q}_2 - 2m_{l2}a_1 l_2 s_2 \dot{q}_1 \dot{q}_2 - \\ -m_{l2}a_1 l_2 s_2 \dot{q}_2^2 + (m_{l1}l_1 + m_{m2}a_1 + m_{l2}a_1)gc_1 + \\ +m_{l2}l_2 g c_{12} \quad (16)$$

$$u_2 = (I_{l2} + m_{l2}(l_2^2 + a_1 l_2 c_2) + k_{r2}I_{m2})\ddot{q}_1 + (I_{l2} + \\ +m_{l2}l_2^2 + k_{r2}^2 I_{m2})\ddot{q}_2 + m_{l2}a_1 l_2 s_2 \dot{q}_1^2 + m_{l2}l_2 g c_{12} \quad (17)$$

where $c_1 = \cos(q_1), c_2 = \cos(q_2), c_{12} = \cos(q_1 + q_2)$ and $s_2 = \sin(q_2)$. For this example we will assume that the two-jointed manipulator is to move from an upward starting position to a horizontal end position as illustrated in Fig. 3. As mentioned, the τ derivatives of q_0 are defined via the time optimal trajectory but with t substituted with τ. For the current example $\tau_f = 0.75$ and $q_0(\tau)$ can be described by three intervals as follows. The first interval is executed at maximum acceleration (4π[rad/s^2]) during $0 \leq \tau < 0.25$, the second at no acceleration but at saturated speed (π[rad/s]) while $0.25 \leq \tau < 0.5$ and at last maximum deceleration when $0.5 \leq \tau \leq 0.75$. This holds for both joints with the exception that joint number one moves in negative angular direction and the second joint in positive. Torque constraints are assumed to be fulfilled for $\tau = t$.

The following settings were used for the dynamic programming algorithm. The number of time steps is set to 30, the grid size 601×56, $\dot{\tau}_{res}$ is two times that of the $\dot{\tau}$-axis resolution as well as t_{max}/t_{min} and $\dot{\tau}_{max}/\dot{\tau}_{min}$ being $3/0$[s] and $1.2/0.01$. Running the algorithm with these parameters took 55[s]. Two optimized sample trajectories with $t_f = 1.5$[s] and 2.25[s] as well as the time optimal trajectory (0.75[s]) are shown in Fig. 4. The energy optimal

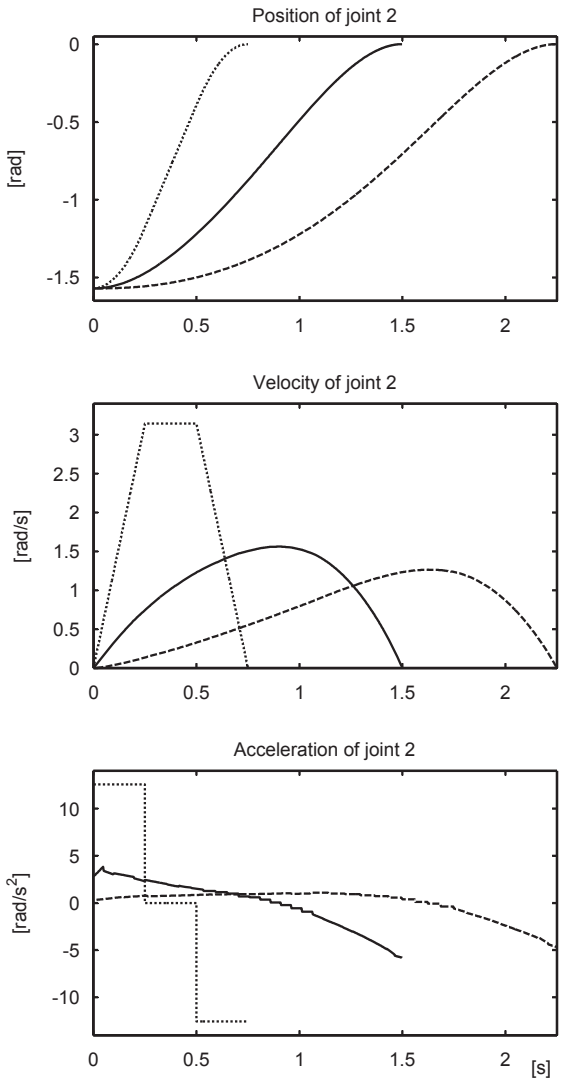

Fig. 4. The dotted line represents the time optimal trajectory while the solid/dashed lines are two energy optimal trajectories with $2/3$ times longer execution time.

trajectories look significantly different even though they are based on the same path. It should be noted that small artifacts can be seen at the starting point of the acceleration. These are due to the lack of a point to point formulation in the dynamic programming model. A higher grid resolution will decrease these. There are also some discontinuities in the acceleration during the middle segments. These are inherent to the second derivative of q_0 being zero at the same time as the first derivative is constant. From equation (4) one can see how a piecewise constant $\ddot{\tau}$ will cause piecewise constant acceleration given these conditions.

Fig. 5 shows a comparison between linear and dynamic time scaling. Energy consumption is plotted as a function of execution time. Already after a 5% increase in execution time, energy consumption is lowered by at least 4 $\%$. When the execution time is more than doubled, the use of energy is decreased even further and continues to be as the available time is increased.

Even though only one run of the algorithm has to be

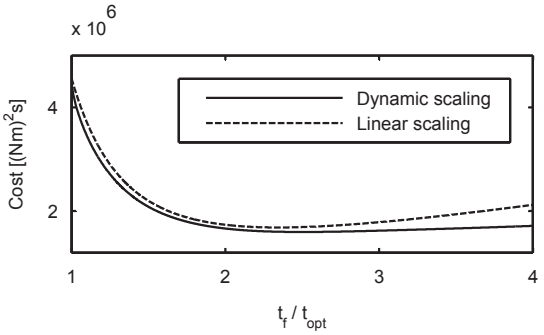

Fig. 5. Comparison of energy consumption when using linear and dynamic time scaling with varying execution time. After a 5% increase in execution time, energy consumption is lowered by at least 4%.

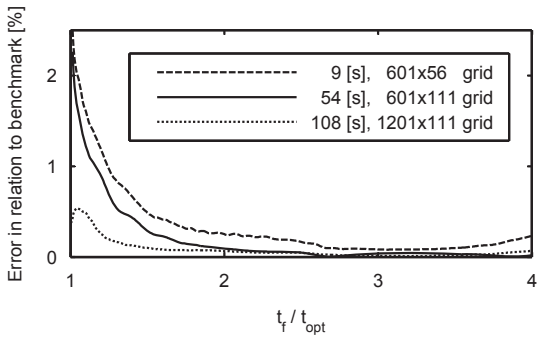

Fig. 6. Error relative to a benchmark for varying settings. The benchmark was performed with the following specs: $6690[s]$ execution time, with a 3001×441 size grid. All runs had $N_\tau = 31$ and $\dot{\tau}_{res}$ at two times the $\dot{\tau}$-axis resolution.

performed, the grid still has to be defined large enough to encompass all sought final time solutions. However, the objective is to find energy functions and not optimal trajectories. Trajectories can be computed later with tailored parameters when a set execution time (t_f^*) has been decided upon. When the approximate $\ddot{\tau}^*$ has been calculated it might be feasible to use a three dimensional grid to generate a trajectory with continuous acceleration. In Fig. 6, the relative error of lower resolution grids compared to one of very high resolution is shown. It can be seen that the $108[\mathrm{s}]$ trial has less than 0.6% error compared to a high resolution benchmark ($6690[\mathrm{s}]$).

V. Discussion and Conclusion

This paper presents a dynamic programming method which can be used to find multiple energy optimal trajectories that follow the same path as a given trajectory. The minimum energy cost for a given execution time can then be used to enhance an existing scheduling algorithm which uses linear time scaling. A two-joint planar manipulator example has been used to investigate the efficiency of the dynamic programming method. Comparing the optimal energy consumption with that of a linear time scaling provides a benchmark for possible improvement. This comparison has shown a minimum energy reduction of 4% for execution times extended with more than 5% compared to the optimal execution time. After a doubling of execution time, the gain from using the energy optimal trajectory is steadily increasing. Concerning the execution of the algorithm we observe that running the algorithm for $108[\mathrm{s}]$ at lower resolution yields less than 0.6% error in relation to a high resolution benchmark ($6690[\mathrm{s}]$). The computational time is low enough for the method to be tractable for generating minimum energy functions $E^*(t_f)$ for each operation in a large system with multiple machines.

The method exhibits some computationally efficient properties. For example, the number of cost function evaluations have been significantly lowered and do not increase with added resolution or maximum value of the t-component in the $t/\dot{\tau}$ grid. Nonuniform gridding can also be used to lower computation time even further, as a high resolution $\dot{\tau}$-axis is only required for long execution times, while t-axis resolution is important for shorter time intervals. For nonuniform gridding of the t-axis, the choice of $t/\dot{\tau}$ as gridded variables is far superior to that of $\tau/\dot{\tau}$ in terms of cost function evaluations. The task is also very easily parallelized, as each time step in the algorithm can be split into multiple threads.

VI. Acknowledgement

This work was carried out at the Wingquist Laboratory VINN Excellence Center within the Area of Advance – Production at Chalmers, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA). The support is gratefully acknowledged.

References

[1] J.E. Bobrow, S. Dubowsky, and J.S. Gibson. Time-Optimal Control of Robotic Manipulators Along Specified Paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.

[2] G. Field and Y. Stepanenko. Iterative dynamic programming: an approach to minimum energy trajectory planning for robotic manipulators. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 3, pages 2755 –2760 vol.3, April 1996.

[3] Alessandro Gasparetto and Vanni Zanotto. A technique for time-jerk optimal planning of robot trajectories. *Robotics and Computer-Integrated Manufacturing*, 24(3):415 – 426, 2008.

[4] M. E. Kahn and B. Roth. The near-minimum-time control of open-loop articulated kinematic chains. *Journal of Dynamic Systems, Measurement, and Control*, 93(3):164–172, 1971.

[5] F.L. Lewis and V.L. Syrmos. *Optimal control*. A Wiley-Interscience publication. J. Wiley, 1995.

[6] B.J. Martin and J.E. Bobrow. Minimum effort motions for open chain manipulators with task-dependent end-effector constraints. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2044 –2049 vol.3, April 1997.

[7] D.S. Naidu. *Optimal control systems*. Electrical engineering textbook series. CRC Press, 2003.

[8] L. Sciavicco, Bruno Siciliano, and B. Sciavicco. *Modelling and Control of Robot Manipulators*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2nd edition, 2000.

[9] Z. Shiller. Time-energy optimal control of articulated systems with geometric path constraints. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 2680 –2685 vol.4, May 1994.

[10] Kang Shin and N. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *Automatic Control, IEEE Transactions on*, 30(6):531 – 541, June 1985.

[11] Kang Shin and N. McKay. A dynamic programming approach to trajectory planning of robotic manipulators. *Automatic Control, IEEE Transactions on*, 31(6):491 – 500, June 1986.

[12] A. Vergnano, C. Thorstensson, B. Lennartson, P. Falkman, M. Pellicciari, Chengyin Yuan, S. Biller, and F. Leali. Embedding detailed robot energy optimization into high-level scheduling. In *Automation Science and Engineering (CASE), 2010 IEEE Conference on*, pages 386 –392, 2010.