High-Level Scheduling of Energy Optimal Trajectories

High level scheduling of energy optimal trajectories

Oskar Wigström, Bengt Lennartson, Alberto Vergnano, Claes Breitholtz

***Abstract*— The reduction of energy consumption is today addressed with great effort in manufacturing industry. A previously presented method for robotic system scheduling, which exploits variable execution time for the individual robot operations, has shown promising results for energy optimization. The method introduces linear time scaling of the trajectories to slow down the manipulators movements. This paper improves the scheduling method by generating energy optimal trajectories 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 in a different execution time frame. With the new method, it is possible to solve the optimization problem for a range of execution times for the individual operations, based on one simulation only. A case study of a cell comprised of four six-link manipulators is presented, in which energy optimal dynamic time scaling is compared to linear time scaling. The results show that a significant decrease in energy consumption can be achieved for any given cycle time.**

***Note to Practitioners*— In robotic manufacturing systems, much energy is wasted due to an adopted minimum time policy for robot operations. This paper presents a method for producing energy efficient operations as well as an optimization model for scheduling these operations in robot cell. Implementation requires a flexible robot controller which allows manipulation of speed and acceleration profiles. The results are also of interest to industrial robot manufacturers, whom have full control over the robot controller and would like to offer energy efficient solutions to their customers.**

***Index Terms*— Automation, Optimization methods**

I. Introduction

Energy efficiency is a very important design driver for robots and other moving devices in manufacturing systems. In fact, oil and electricity prices are rapidly increasing, and also companies will have to undergo new environmental policies for a sustainable future. This can be accomplished by research on new efficient equipment, with for instance regenerative braking, energy storages and lightweight solutions. As a short time strategy however, this paper focuses on existing hardware solutions and presents a method for the optimization of their usage.

The final performances of a robotic manufacturing system result from the synergistic interaction between mechanics, control and software [1]. Thus, the optimization of such systems should exploit different research fields. Energy optimization of mechatronic devices is well investigated in [2], [3], [4], [5]. Energy optimal trajectories for robot applications is a big research field itself, see e.g. [6], [7] and [8]. From a system design perspective, a selection and matching of efficient design solutions for pre-defined operations is studied in [9], [10], [11]. The optimization of the scheduling supervisor of the overall system is a promising area, although most works are concerned with cycle time, as in [12], [13]. Two approaches are presented in [14], where idle time between the operations is used to reduce velocities and accelerations, without concern to the energy consumption.

A method for decreasing the energy consumption in production systems was presented [15]. Its assumption is that a minimal cycle time for a manufacturing sequence does not imply that every operation has to be performed at maximum speed. This is the case when two robots work in a shared zone for some operations, and one must wait for the other to unbook the zone, or when a bottleneck robot constrains the cycle time of others in series or parallel. In these situations it is demonstrated that it might not be convenient to run all operations at maximum speed, and neither to slow down the operations to use all the available waiting time. In fact, in general, there is an energy optimal execution time which depends on the specific robot hardware and the programmed operation. One of the building blocks in the optimization method is the calculation of the energy consumption $E_\ell(t_{f\ell})$ for each individual robot operation ℓ as a pseudo-polynomial function of its execution time $t_{f\ell}$. These local energy functions are then used to determine the globally optimal execution times $t^*_{f\ell}$ for all operations, such that the overall energy consumption of the system is minimized. The energy functions in [15] are based on a uniform time scaling of the robot trajectory, i.e. a linear stretching of the position profile, providing promising results. In this paper we improve the method in [15] introducing a dynamic time scaling, in order to deliver an energy optimal trajectory for the operations in a manufacturing sequence.

In [16] we extended the aforementioned method by using dynamic programming in order to solve the trajectory planning problem for a range of execution times simultaneously. Preliminary results for dynamically scaled operations were also presented for a two-joint planar robot arm following a simple trajectory. A comparison between dynamic and linear scaling showed that if the operations execution time was extended by more than 5%, dynamic scaling would use at least 4% less energy than linear scaling. In this paper, dynamic programming for generation of optimal energy functions

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) and the Swedish Research Council. The support is gratefully acknowledged.

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

A. Vergnano, Department of Mechanical and Civil Engineering, University of Modena and Reggio Emilia, Modena, Italy

is combined with high-level scheduling. The scheduling problem is, as in [15], modeled with mixed integer linear constraits, with a nonlinear cost function expressing the energy consumption of the system. A case study with four industrial robots operating in the same cell is also reported.

The paper is structured as follows. Section II contains an introduction to trajectory planning as well as the optimization model and the algorithm used to solve it. Section III briefly covers the scheduling problem. Section IV presents the results from a four robot test case and finally in Section V conclusions are drawn along with a brief discussion.

II. Trajectory Planning

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 [17], 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 [18], [19] 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 [20]. In summary, after minimum time, focus shifted towards minimum energy, which produced smoother trajectories and smaller tracking errors.

There are a large number of approaches to the minimum energy problem. Solutions include dynamic programming [21], and later iterative dynamic programming [22], parameterized b-splines [23], Pontryagins maximum principle [24], among others, were used. Some of the later methods allow for constraints on the jerk and result in a continuous acceleration. In some cases a simplified cost function is proposed, for example only minimizing the squared join 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 of 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 suggested in [21], 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 [21], which employs a weighting between cost and time, our method includes elapsed time in the optimization model. This implies that, while [21] 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 the plant model complexity or objective function. 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 [21] is somewhat lower than ours, the dynamic programming optimization will have to be run once for every execution time $t_{f\ell}$ of interest in the energy function, $E^*_\ell(t_{f\ell})$. 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 is based on a single scaling parameter, the dimensionality will be unchanged for additional robot joints as well as more intricate energy models.

A. 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 [25], pp. 131-140. Einstein summation convention is used, as in [21], 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 is the inertia matrix, C the tensor of centrifugal and Coriolis coefficients, F the viscous friction matrix, G the gravitational vector and q^i the angular position of joint i. Note that J, C and G are all functions of q. 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 and its derivatives. This implies that τ is the time scale for the time optimal trajectory, q_0. For example, defining $\tau = t$ would result in the time optimal trajectory. Modeling tools such as ABB RobotStudio [26] can be used to generate q_0. The relationship between q and q_0 can be expressed as

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

where $\tau(t)$ is a monotonically increasing function with a starting value of 0 and final value τ_f, where τ_f in our case corresponds to the time optimal execution time. If $\tau(t_f) = \tau_f$, then t_f is the new final execution time of the dynamically scaled operation. 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,

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

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

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

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

The optimization procedure is also subject to a number of dynamic constraints, such as limits on torque, acceleration and speed. This can be implemented using a barrier function, or since the original trajectory is assumed to be time optimal, by adding the constraint $\dot{\tau} <= 1$.

Considering the operation execution times relevant to this paper, as in [7] and [27] , the energy consumption of an AC permanently excited synchronous motor can be expressed by the following simplified voltage and current models

$$\begin{aligned} V_i(t) &= R_iI_i(t) + K_{V,i}K_{R,i}\dot{q}_i(t) \\ I_i(t) &= T_i(t)/(K_{T,i}K_{R,i}) \end{aligned} \quad (6)$$

where $I_i(t)$ and $V_i(t)$ are the equivalent DC current and voltage of the i:th rotor, R_i is the stator resistance, $K_{V,i}$ is the electrical (back emf) constant, $K_{R,i}$ is the transmission gear ratio and $K_{T,i}$ is the equivalent torque constant. With (6) defined, we can express the power of each motor as

$$P_i(t) = V_i(t)I_i(t) = \frac{R_i}{K_{T,i}^2K_{R,i}^2}T_i^2(t) + \frac{K_{V,i}}{K_{T,i}}T_i(t)\dot{q}_i(t) \quad (7)$$

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. Integrating (7) gives an expression for the total energy consumption, the cost that is to be minimized

$$E(t_f) = \int_0^{t_f}\left(\sum_{i=1}^{n}P_i(t)\right)dt \quad (8)$$

Here, n is the number of joints. With the power and torque defined as in (7) and (5), the cost E is a functional of q_0 and τ. Since the former is known, solving the optimization problem is a matter of 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.

B. Optimization model

As mentioned, solving the trajectory planning problem entails finding the τ that minimizes a given cost function. Define the second derivative of τ as

$$\ddot{\tau}(t) = u(t), \quad (9)$$

where $u(t)$ is a control input. Also, introduce a time-varying sampling time h_k that affects the time updates as

$$t_{k+1} = t_k + h_k \quad (10)$$

and let the input variable be peicewise constant during the sampling intervals, i.e.

$$u(t) = u(t_k), \qquad t_k \le t < t_{k+1}, \quad (11)$$

The decision to use a piece-wise 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 (4), these minor artifacts can be considered marginal. One could instead choose to define the 3:rd derivative of τ as constant, and instead achieve only discontinuous jerk, but at the cost of dimensionality and complexity. Discretization of (9), with a sampling period h_k and constant control input as in (10) and (11), gives the discrete state space model

$$\begin{bmatrix} \tau_{k+1} \\ \nu_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h_k \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \tau_k \\ \nu_k \end{bmatrix} + \begin{bmatrix} h_k^2/2 \\ h_k \end{bmatrix}u(t_k) \quad (12)$$

where $\nu = \dot{\tau}$ and for simplicity, we introduce $\tau_{k+1} = \tau(t_{k+1})$, $\tau_k = \tau(t_k)$ and $\nu_{k+1} = \nu(t_{k+1})$, $\nu_k = \nu(t_k)$. The minimization of (8), including this discrete time model of the time function $\tau(t)$ can be solved with dynamic programming, but for computational reasons discussed later, it is convenient to reformulate the problem. Since τ is monotonically increasing it is possible, instead of taking steps along the t-axis in each iteration, to take steps along the τ-axis and let (10) act as a discrete state equation. Define

$$h_k\nu_k + h_k^2u(t_k)/2 = \Delta_k, \quad (13)$$

where Δ can be regarded as a user defined sampling period or gridding of τ. If (13) is inserted into (12), then in every step k, τ will be updated as

$$\tau_{k+1} = \tau_k + \Delta_k \quad (14)$$

Equation (13) can also be manipulated into an expression for the control signal, $u(t_k) = 2(\Delta_k - h_k\nu_k)/h_k^2$. Inserting this expression for $u(t_k)$ into the bottom equation of (12) gives a new state equation for ν. Regarding the sampling time, h_k as the new control signal and letting (10) act as a state space equation leads us to the reformulated discrete state space model

$$\begin{bmatrix} t_{k+1} \\ \nu_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} t_k \\ \nu_k \end{bmatrix} + \begin{bmatrix} h_k \\ 2\Delta_k/h_k \end{bmatrix} \quad (15)$$

From here, dynamic programming can be applied to solve the discrete optimal control problem. The relation between (14) and (10) can be seen as a mapping of τ onto t, which is illustrated in Fig. 1.

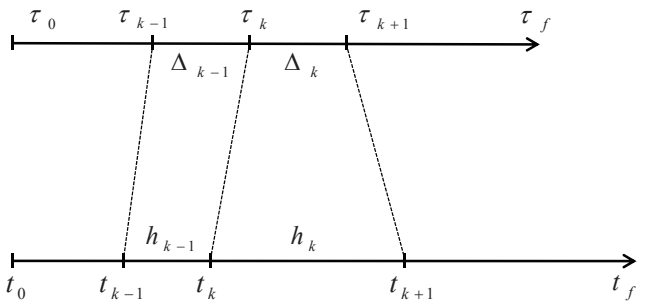

Fig. 1. Uniformly spaced τ mapped onto t.

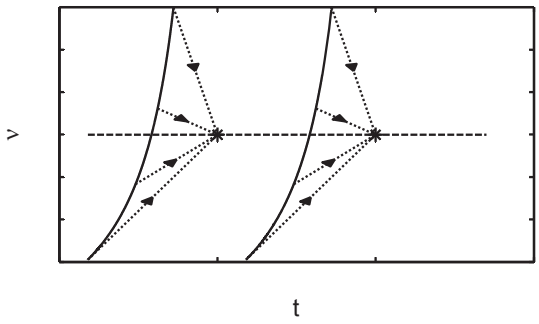

Fig. 2. The solid line denotes (t_k, ν_k) and the two points show (t_{k+1}, ν_{k+1}). Note that the solid line has the same form for all k. The dashed line represents a constant $\nu_{k+1} = 0.6$. The dotted lines illustrates that the point (t_{k+1}, ν_{k+1}) is reachable from (t_k, ν_k).

C. Dynamic programming

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 process represents a movement within this grid, either from point to point or, where this type of formulation is inconvenient or not possible, from a location in between grid points where a specific 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 [28] or [29].

Solving the optimal control problem for the system (15) and the cost (8), i.e. finding the optimal h_k $k = 1, ...N_\tau$ and ν_0, requires the two variables t and ν to be represented by a discrete grid. The number of elements along each axis in the grid is defined by N_t and N_ν. If the optimal cost for a location in this grid for a specific k is denoted as $J_k^*(t_k, \nu_k)$, then the functional equation of dynamic programming in the forward direction can be written as

$$J_{k+1}^*(t_{k+1}, \nu_{k+1}) = \min_{h_k}[V_k(t_k, \nu_k, h_k) + \ J_k^*(t_k, \nu_k)] \tag{16}$$

Here, V is the cost of moving from (t_k, ν_k) to (t_{k+1}, ν_{k+1}), i.e. the cost in (8), but with the integration from t_k to t_{k+1}. Then, for each (t_{k+1}, ν_{k+1}) point in the grid, compute J_{k+1}^*. Given the system described by (15), there is only one degree of freedom in J_{k+1}^* which is h_k. Even though the possible solutions may not coincide in specific (t_k, ν_k) grid points, it is still possible to interpolate values from J_k^*. Solving (16) for all k will result in the optimal cost for all the points in the grid. At the final iteration N_τ, the resulting matrix $J_{N_\tau}^*$ will hold the minimum cost for every grid point. Since t_k represents elapsed time, $t_{N_\tau} = t_f$, and as such the minimum cost for all execution times within the range of t_{N_τ} can be found based on the matrix $J_{N_\tau}^*$.

Before starting the optimization, the first and second derivative of q_0 with respect to τ are required. 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}, ν_{k+1}) in the grid having a corresponding (t_k, ν_k) curve from where that point can be accessed at step $k + 1$. This movement in the grid is illustrated in Fig. 2. The minimization part in (16) 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 from specific grid points, be reminded that the cost will have to be interpolated.

D. Algorithm

The algorithm works as follows, define a data 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 t_k, h_k, ν_k and $c1$ at lines 6, 7 and 9 in the routine OPTIMALSOURCE are column vectors. The matrix multiplication will thus clone the 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 J(:, :, k + 1) ←
 5        OPTIMALSOURCE(k, J(:, :, k), opt)
 6  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, call OPTIMALSOURCE and save the results to $J(:, :, k + 1)$. This function will return

the optimal value J^*_{k+1} for all points (t, ν) in the grid. This action is repeated N_τ number of times, i.e. the number of steps that should be taken along τ. 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 $J(t, :, N_\tau)$. To find the optimal trajectory for a specific execution time, J can be used to trace the optimal path through the grid.

The OPTIMALSOURCE method works as follows. For each ν_{k+1} in the grid, N_ν number of times, the following is performed. Generate the possible relative grid points from where ν_{k+1} is reachable. Use these points to compute the cost to go. At row 5, compute which time indices in the grid that are relevant, i.e. discard values that cannot be reached. On rows $6-7$, two matrices are generated that contain the t_k and ν_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. The summation on row 10 corresponds to that of V_k and J^*_k in (16), and choosing the minimum element relates to the minimization argument.These are the optimal costs for all t_{k+1} along the current ν_{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 5. This result is then saved to $J(:, e, k+1)$.

OPTIMALSOURCE$(k, J(:, :, k), opt)$
1 **for** $e \leftarrow 1$ **to** N_ν
2 **do**
3 $[\nu_k, h_k] \leftarrow$ GRIDPOINTS$(\nu_{k+1}(e), opt)$
4 $c1 \leftarrow$ COSTFUNCTION(ν_k, h_k, k, opt)
5 $t_{k+1} \leftarrow$ GETTIMEINDICES$(J(:, :, k), opt)$
6 $T_k \leftarrow t_{k+1} * ones(1, N_\nu) - (h_k * ones(1, N_t))^T$
7 $X_2 \leftarrow (\nu_k * ones(1, N_t))^T$
8 $C2 \leftarrow$ INTERPOLATE$(J(:, :, k), X_2, T_k, opt)$
9 $C1 \leftarrow (c1 * ones(1, N_t))^T$
10 $C \leftarrow C1 + C2$
11 $f \leftarrow$ MIN(C, row)
12 $J(:, e, k+1) \leftarrow$ FILLUP(f, t_k, opt)
13 **return** $J(:, :, k+1)$

E. Complexity

The shape of the (t_k, ν_k) curve, as shown in Fig. 2, does not change for different t_{k+1} along a constant ν_{k+1}, because (9) is time invariant. Since the scaling variable definition is time invariant, the cost function (8) is also time invariant, thus the cost function for a specific ν_{k+1} is a function of h_k and ν_k. This implies that the cost function does not have to be evaluated separately for each t_{k+1} where the value of ν_{k+1} is the same. Note how cost function in the OPTIMALSOURCE function at row 4, only evaluates a vector with N_ν elements. At row 8 however, interpolation is performed for a matrix of size $N_t \times N_\nu$. If instead, the original discrete system (12) had been used, each point in the τ/ν-grid would have to be evaluated separately. Since each iteration k is time invariant, the cost evaluations from previous iterations can be reused.

Using the reformulated method in (15), a total number of $(N_\tau - 1)$ iterations will be performed, and for each iteration, N_ν points are connected to another N_ν points. As previously mentioned, the number of cost function evaluations does not increase with the size of N_t, while the number of interpolations from the grid increases by a factor N_ν. Thus, the number of cost function evaluations is relative to $(N_\tau - 1)N_\nu^2$ while the number of interpolations from the grid is relative to $N_t(N_\tau - 1)N_\nu^2$. Note that cost function evaluations are much more costly than interpolations.

For the original model (12), allowing reuse of cost function computations, the same expressions for cost function evaluations and interpolations are derived. But the size of N_τ and N_t are not the same as for the reformulated model (15). In the (12) case, t is a discrete variable and τ is a discrete approximation of a continuous variable. Thus the resolution of τ must be higher than that of t. For (15) however, t is a discrete approximation, τ is discrete and the resolution of t needs to be finer than that of τ. From experience, a reasonable accuracy is achieved by an at least 5 times higher resolution for the discrete approximation. As such, (12) requires a minimum of 5 times as many cost function evaluations as (15).

Also, if one would like to minimize the computational requirements, using a nonuniform grid is a good idea. For long execution times, the value of h_k for each iteration will larger, thus the resolution of the t-axis can be lower for its upper range. On the other hand, long execution times imply small values and changes in ν, therefore a higher resolution is preferable at the lower range of ν. But if a variable resolution on t is used and one takes steps of varied length along the t-axis for each iteration, as in the original model (12), one cannot reuse cost function evaluations each iteration. This means that the additional performance gained from nonuniform gridding cannot be utilized by (12).

As previously mentioned, it would also be possible to use the model from [21] and solve it multiple times iterating the weighted cost function each time. What follows is a rough comparison.

To solve ONE trajectory planning instance of [21], the algorithm must check $(N_\tau - 1)$ times if each point in the ν grid (N_ν) is connected to any other ν points (N_ν). Thus it has a time dependency relative to that of $(N_\tau - 1)N_\nu^2$. Note that we want to solve multiple instances, the number of instances is relative to that of the longest time scaling. The computational requirement for generating a complete energy function is therefore of order $N_i(N_\tau - 1)N_\nu^2$, where N_i is the number of executions needed to map the energy function.

It is reasonable to assume that the reformulated model (15) compared to the original model [21] will use the same size of N_τ while N_ν is expected to be larger. The magnitude of the latter is dependent on the range of final times that are of interest. For our experiments, a time scaling of 5 times the original execution time was well enough, most energy functions would increase asymptotically with time at longer

execution times. We believe that for a 5 times scaling, N_ν does not need to be larger than twice the size of N_ν for [21], while producing results at the same resolution. This suggests that the number of cost function evaluations for the reformulated model (15) in this paper is about 4 times as many compared to [21]. Also considering the additional interpolations required for our algorithm adds approximately 30% computational time. In total this implies approximately 5 times longer computation time for our method, solving one instance. However, 5 executions of [21] ($N_i = 5$) are not enough to reach a sufficient resolution for the sought energy functions, thus making the reformulated model (15) in this paper more efficient for the generation of energy functions.

III. Energy Optimal Scheduling

With the energy function for each operation generated, an energy optimal schedule can be derived for the given process. The constraints governing the scheduling problem can be expressed with linear constraints consisting of real and binary variables. For a thorough account on mixed integer constraint modeling, see for example [30] or [31]. The decision variables for the problem are the real valued starting and stopping times for each robot operation as well as binary variables representing mutual exclusion. The energy function data from the dynamic programming algorithm are approximated as polynomials. These polynomials are checked for convexity and that the relative error related to the original data is within appropriate bounds. The cost function for the scheduling problem can now be formed using the polynomials. These polynomials make it easy to specify an analytical gradient and hessian for the solver. As such, the problem class is that of a convex Mixed Integer Nonlinear Linear Program (MINLP).

The MINLP master problem can be divided into a number of subproblems, one for each ordering combination. These are enumerated and each subproblem is treated as a linearly constrained problem with a nonlinear cost function. Not all of these choices are feasible, depending on the final time specified for the model. After checking for feasibility, the valid subproblems are solved using MATLAB's optimization toolbox. The algorithm employed uses an interior point method combined with a barrier function for the constraints. The interior point method is described in [32], [33] and [34]. There are of course more efficient approaches to the master problem than explicit enumeration. However, as the primary focus of this paper is that of energy optimal trajectories and their impact on the scheduling solution, implementing advanced scheduling techniques has not been our concern. There are of course many modern techniques for solving convex MINLP including among others: Branch-and-Bound, Outer-Approximation, LP/NLP-based branch and bound [35], [36].

A. Constraint modeling

Let the global starting and finishing time for the j:th operation executed by the i:th robot be denoted t_{ij}^s and t_{ij}^f. A sequence of two operations is simply expressed by defining the finishing time of the preceding operation as smaller than the starting time of the following operation,

$$t_{ij}^s \geq t_{lm}^f + \epsilon, \tag{17}$$

where l and m is the robot and operation index of the first operation, i and j that of the second and ϵ a significantly small positive constant. It is also required to limit the minimum execution time of operations. If an operation j in robot i has a minimum execution time of $T_{0,ij}$, then the execution time can be constrained by

$$t_{ij}^f \geq t_{ij}^s + T_{0,ij} \tag{18}$$

Shared resources can be expressed in the following way. If two operations, ij and lm share the same resource, then define $\sigma_{ij,lm}$ as a boolean variable representing operation ij being performed after lm. Also $\sigma_{lm,ij}$ is a boolean variable representing the negation of the previous statement, i.e. operation lm is performed after ij. The resulting constraints for this example are

$$\begin{aligned} t_{ij}^s &\geq t_{lm}^f - M(1 - \sigma_{ij,lm}) \\ t_{lm}^s &\geq t_{ij}^f - M(1 - \sigma_{lm,ij}) \\ \sigma_{ij,lm} + \sigma_{lm,ij} &= 1 \end{aligned} \tag{19}$$

In other words, a boolean variable or expression is used to negate constraints when false. As such, the constant M needs to be sufficiently large for this negation to be valid. The same principle can be used for one robot having an alternative order of execution for its operations.

For constraining the cycle time, an additional variable is added. This variable is constrained to be larger than the stopping time of the last operation for each robot. Adding an upper bound for the new variable will now constraint the complete cycle time of the cell. With the scheduling problem described by these mixed integer linear constraints and the cost function by the convex polynomials, the optimization model can no be solved using a any standard MINLP solver.

IV. Case Study

For the case study, an example with four industrial robots is considered. The four robots work together on a single work piece located in between the robots. There is a total number of 40 operations, all for which an energy function, $E(t_f)$, is computed. The cell includes three common zones in which only one robot can work at a time. Each robot has a three tasks to perform including between one and five operations, most of which belong to common zones. Also, one of the tasks for each robot requires the other two tasks to be completed in order to be allowed to start. The scheduling problem has 82 real and 32 binary variables, and in total there are 1280 valid integer combinations. For a near time optimal cycle time, only 44 combinations were feasible. Note that if a robot is not performing any operation, there is still a cost based on the gravity term in (1), this is called the holding power. It can be added to the cost function just like the energy functions. This time however only a first degree

expression is needed as the holding power increases linearly with time.

The robot cell was modeled with ABB RobotStudio from where trajectory information for each operation was extracted. All optimization was run on a Windows 7 64bit system with a 2.66 [Ghz] Intel Core2 Quad CPU and 4 [GB] of RAM. The final time feasibility check for each subproblem, which also generates a starting point, took < 0.1 [s] in 98% of the cases. Most subproblems were solved in less than $10[s]$. It should be noted that for a few instances the initial barrier function weighting had to be varied in order for MATLAB to produce an optimal solution. As for the minimum energy trajectory planning problem, with $N_\tau = 30$, $N_\nu = 50$ and $N_t = 186$, each operation instance was solved in close to 40 [s].

A. Results

A sample operation has been picked to illustrate the properties of dynamic scaling on individual operations. Fig. 3 illustrates the energy consumption profiles for three various execution times. The dotted curve belongs to the time optimal trajectory with an execution time of roughly 10 [s], the solid and dashed are the same trajectory but with the final time extended by factor a 1.75 and 2.75. Note how the first part of the curves has converged for the two scaled cases. This reduces the time optimal trajectory's characteristically initial high acceleration. The dashed curve has an almost constant power consumption between 5-15[s], this is because during this segment of the path, the gravitational force affecting the manipulator is the lowest. Because of this the optimization results in a slow traversal of this segment, minimizing the cost of gravity. In Fig. 4 the resulting energy function for the sample operation is presented. The dashed line, which increases linearly, shows the time optimal energy cost and the holding power required to stay stationary for the remaining time. The dotted and solid curves are the energy cost for linear and dynamic scaling. Note that not all operations differ that much between linear and dynamic scaling, the average difference should be somewhat apparent from the overall result in Fig. 5. All the operations scaled resulted in convex functions. This was to be expected as regenerative breaking was not considered.

In Fig. 5, the total energy consumption for all four robots is shown, running an energy optimal schedule. The dashed line shows the result of no scaling, i.e. minimum energy scheduling is performed but no scaling is allowed. The dotted and solid curves represent linear and dynamic scaling. While upholding a time optimal cycle time, linear scaling reduces the energy usage by 11%, and dynamic scaling a total of 18%. If the cycle time is allowed to be extended by 10% (10[s]), linear scaling will reduce energy cost by 18%, and dynamic scaling as much as 28%.

For the sample operation considered in Fig. 3, an analysis of the Root Mean Square Error (RMSE) for varying parameter values can be found in Fig. 6. Top dotted curve shows variations in N_ν. For the solid, the resolution of τ is varied. Note that this also changes the resolution of t,

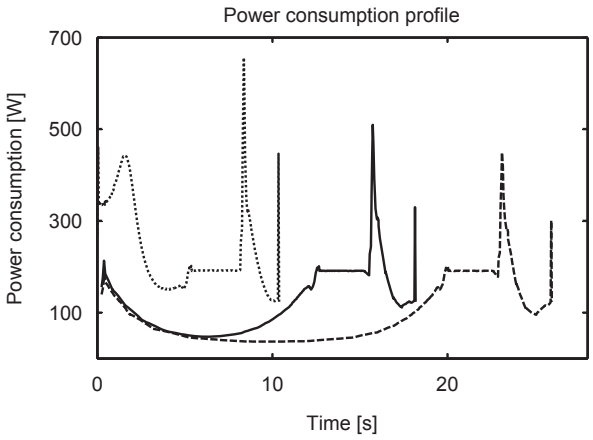

Fig. 3. The power consumption profiles for a sample operation of one six DOF industrial robot with three various execution times. The dotted curve belongs to the time optimal, the solid and dashed curves are the same trajectory but with the final time extended by factor 1.75 and 2.75.

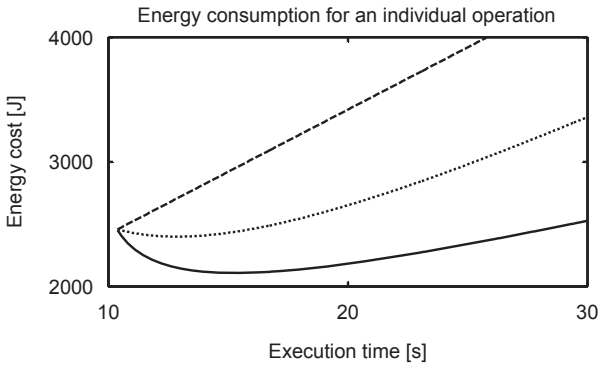

Fig. 4. Three energy functions based on the same sample operation as in Fig. 3. The dashed line is based on time optimal execution and holding power required to stay stationary for the remaining time. The dotted and solid curves are the minimal energy costs for linear and dynamic scaling.

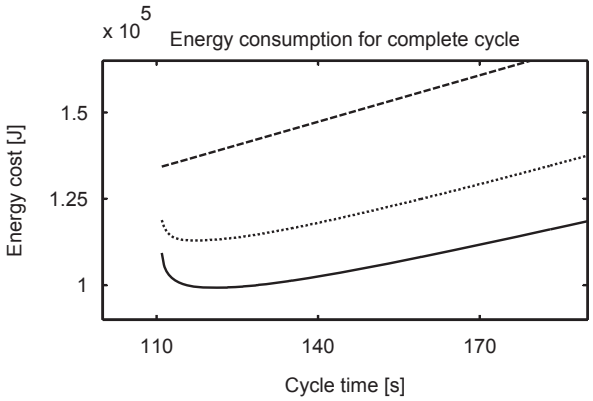

Fig. 5. Overall minimum energy consumption for the case study as a function of cycle time. The dashed line shows scheduling where no scaling is allowed, the dotted and solid curves represent scheduling based on linearly and dynamically scaled operations.

$N_t = 6.2/N_\tau$. The dashed curve shows a varying N_t. All tests were performed for a 5 times longer execution time than the time optimal. For all three parameters, higher resolution decreases the RMSE. The error tends to zero for all but the dotted curve which stabilizes in the range of 0.1% RMSE. This can be contributed to numerical errors. We conclude that it is possible to run the algorithm at low resolutions with only relatively small errors. We estimate that the settings used for the case study, mentioned eariler, should not produce an error of more than 1% RMSE. In fact, N_ν could actually be lowered to about 20 without any significant impact.

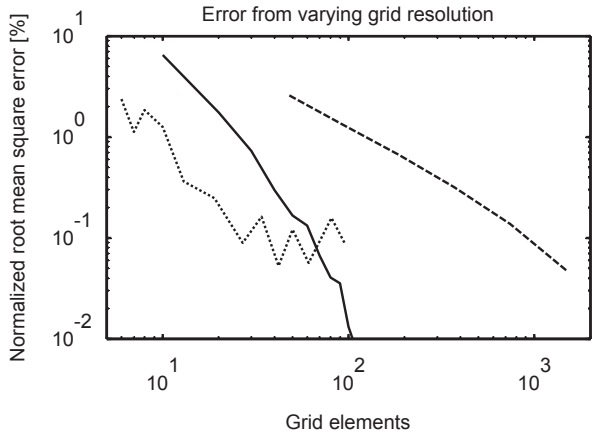

Fig. 6. The resulting normalized root mean square error for varied parameter values compared to a benchmark. The dotted cruve is varied N_ν, with benchmark $N_\tau = 30$, $N_\nu = 119$, $N_t = 186$. The solid curve shows varied N_τ and $N_t = 6.2N_\tau$, benchmark $N_\tau = 120$, $N_\nu = 96$, $N_t = 744$. Last, the dashed curve is varied N_t, benchmark $N_\tau = 30$, $N_\nu = 50$, $N_t = 2961$.

V. Discussion and Conclusion

This paper presents a dynamic programming method, which can be used to find multiple energy optimal trajectories with varying execution times that follow the same path as a given trajectory. The minimum energy cost for a given execution time can then be used for scheduling of multiple robots working in the same environment. The modification of the original trajectory is defined as a dynamic scaling. Previously, linear scaling of operations has been used for scheduling. A case study of four six-joint industrial robots with booking of common zones is presented in order to evaluate the possibility of energy reduction using this method. Results show that linear scaling can decrease the total energy cost by 10-20%. Employing the dynamic scaling, as suggested in this paper, will reduce the energy cost by an additional 10%. Even though energy consumption for smaller robots can be considered marginal, larger industrial robots carrying heavy loads and other automated robotic machinery used for heavy lifting can benefit greatly from the suggested scheduled energy optimal trajectories.

References

[1] R. Isermann. *Mechatronic Systems: Fundamentals*. Springer, 2005.

[2] R. Saidur. A review on electrical motors energy use and energy savings. *Renewable and Sustainable Energy Reviews*, 14(3):877 – 898, 2010.

[3] R. Visinka. *Ch. 2 - Energy Efcient Three-Phase AC Motor Drives for Appliance and Industrial Applications*. Goldberg and Middleton, 2002.

[4] A. Yang, J. Pu, C.B. Wong, and P. Moore. By-pass valve control to improve energy efficiency of pneumatic drive system. *Control Engineering Practice*, 17(6):623 – 628, 2009.

[5] G. Hirzinger, N. Sporer, A. Albu-Schaffer, M. Hahnle, R. Krenn, A. Pascucci, and M. Schedl. Dlr's torque-controlled light weight robot iii-are we reaching the technological limits now? In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1710 – 1716 vol.2, 2002.

[6] H. Diken. Energy efficient sinusoidal path planning of robot manipulators. *Mechanism and Machine Theory*, 29(6):785 – 792, 1994.

[7] J. S. Park. Motion profile planning of repetitive point-to-point control for maximum energy conversion efficiency under acceleration conditions. *Mechatronics*, 6(6):649 – 663, 1996.

[8] E.S. Sergaki, G.S. Stavrakakis, and A.D. Pouliezos. Optimal robot speed trajectory by minimization of the actuator motor electromechanical losses. *J. Intell. Robotics Syst.*, 33:187–207, Feb. 2002.

[9] O. Maimon, E. Profeta, and S. Singer. Energy analysis of robot task motions. *Journal of Intelligent and Robotic Systems*, 4:175–198, 1991. 10.1007/BF00440418.

[10] Fredrik Roos, Hans Johansson, and Jan Wikander. Optimal selection of motor and gearhead in mechatronic applications. *Mechatronics*, 16(1):63 – 72, 2006.

[11] T. Izumi, H. Zhou, and Z. Li. Optimal design of gear ratios and offset for energy conservation of an articulated manipulator. *Automation Science and Engineering, IEEE Transactions on*, 6(3):551 –557, Jul. 2009.

[12] S. Panek, O. Stursberg, and S. Engell. Optimization of timed automata models using mixed-integer programming. In *In Formal Modeling And Analysis of Timed Systems, volume 2791 of LNCS*, pages 73–87. Springer, 2004.

[13] A. Kobetski and M. Fabian. Time-optimal coordination of flexible manufacturing systems using deterministic finite automata and mixed integer linear programming. *Discrete Event Dynamic Systems*, 19:287–315, Sep. 2009.

[14] A. Kobetski and M. Fabian. Velocity balancing in flexible manufacturing systems. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 358 –363, may 2008.

[15] 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.

[16] O. Wigström and B Lennartson. Energy optimization of trajectories for high level scheduling. In *Automation Science and Engineering (CASE), 2011 IEEE Conference on*, 2011.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] "ABB RobotStudio" Internet: http://www.robotstudio.com, [Aug. 1, 2010].

[27] D.Y. Ohm. (2006, feb. 3). Selection of servo motors and drives (rev.2)[Online]. Available: http://www.drivetechinc.com.

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

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

[30] Y. Pochet and L.A. Wolsey. *Production Planning by Mixed Integer Programming (Springer Series in Operations Research and Financial Engineering)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[31] H. P. Williams. *Model Building in Mathematical Programming, 4th Edition*. Wiley, 4 edition, October 1999.

[32] R.H. Byrd, J.C. Gilbert, and J Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89:149–185, 2000. 10.1007/PL00011391.

[33] R.H. Byrd, M.E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9:877–900, 1997.

[34] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Math. Program.*, 107:391–408, Jul. 2006.

[35] F.P. Fernandes, M.F.P. Costa, and E.M.G.P. Fernandes. Overview on mixed integer nonlinear programming problems. *AIP Conference Proceedings*, 1168(1):1374–1377, 2009.

[36] S. Leyffer, J. Linderoth, J. Luedtke, A. Miller, and T. Munson. Applications and algorithms for mixed integer nonlinear programming. *Journal of Physics: Conference Series*, 180(1):012014, 2009.