Proceedings of the 2009 IEEE International Conference on Mechatronics.
Málaga, Spain, April 2009.

Throwing motion generation using nonlinear optimization on a 6-degree-of-freedom robot manipulator

Ferenc Lombai[1]
[1]Faculty of Information Technology
Péter Pázmány Catholic University
Práter u. 50/A, H-1083 Budapest, Hungary
Email: lomfe@digitus.itk.ppke.hu, Tel: +3618864771
Fax: +3618864725

Gábor Szederkényi[1,2]
[2]Systems and Control Laboratory
Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende u. 13-17, H-1111 Budapest, Hungary
Email: szeder@scl.sztaki.hu, Tel: +3612796163
Fax: +3614667503

***Abstract*—A 6-degree-of-freedom rigid robot arm and its throwing motion generation is described in this paper. The trajectories for the joint variables are generated off-line as a cubic spline obtained using general constrained nonlinear optimization, taking into consideration limitations (position, speed, acceleration and jerk) of the joint actuators, and the current limit of the whole structure. The obtained trajectories are previously checked to avoid collisions using oriented bounding boxes and their separating axis theorem tests. The trajectory tracking of the individual joint is done using a discrete-time constrained optimal control technique.**

I. INTRODUCTION

The capability of throwing was a great step in the human evolution and the reproduction of throwing motions with a robot manipulator is undoubtedly a challenging problem. The interesting phenomena behind the human throwing is, that people are capable to aim so close to a point far away, that the release moment of their motion have to be chosen more precisely, than that the time constant of a neuron (typically above $1ms$) would allow. Werner Heisenberg mentioned this phenomena in [1] that also implies the necessity of some precognition for such movements. This concept can be implemented as a knowledge base of preprocessed motion tasks, which can be used by higher level control methods that will be the subject of further work.

Relatively few publications report results in the field of throwing and almost all of them deal with lower degree-of-freedom robot arms. In [2], the generation of a throwing motion for a 2 DOF robot is described. The trajectory planning takes into consideration torque, angular velocity and time limits, while the controller includes a nonlinear dynamic compensator and a parameter estimator to choose the best time instant for the release of the thrown object. Among other complex tasks (e.g. catching, batting and pushing), motion planning experiments for throwing without grasping for low degree-of-freedom robots are described in [3].

In this paper, we give a possible solution for the above mentioned motion planning task by the use of general constrained nonlinear optimization as many other practical robotic solutions do (see e.g. [4]–[6]). For this, we present an extended implementation of the algorithm [7] for planning time-jerk optimal trajectories on a 6-DOF robot arm. The original method has been significantly reworked to meet the specific needs of our problem. Beyond the modified optimization space and objective function, additional constraints involve the nonlinear optimization related to joint variable limits and to the instantaneous power consumption of the actuating motors.

The solution of the above mentioned problems are now supported with effective software tools such as Matlab® and its Optimization Toolbox or the Mathematica environment for numeric and symbolic computations.

The structure of the paper is the following. In section II the problem is formulated and the assumptions are described, section III contains the brief overview of the experimental system's hardware and software components. The description of the algorithm can be found in section IV while section V contains the applied trajectory tracking control scheme. The most important simulation and measurement related results are shown in section VI. Finally, a summary concludes the paper in section VII.

II. DESCRIPTION OF THE PROBLEM

There are numerous conditions that influence the attributes of a throwing motion but there are similarly many possible metrics that can be constructed to measure the quality of those motions. The most important motivation behind this work is to generate possible motions and using the results, investigate the dynamic capabilities of the robot arm. This process can be considered as an optimization process in a high dimensional space fragmented by strict nonlinear constraints. The problem is further complicated by the well known fact that the computation of the necessary inverse kinematic functions are often produce irrelevant results or simply no solution exists. Moreover, numerical instabilities can occur due to the singularities in the jacobian too. Since the constrained nonlinear optimization method that we will use is gradient based, it is necessary to formulate the constraint and objective functions (that are defined over a bounded field) as smooth as possible.

Firstly, it is necessary to define a space in which each point characterizes one throwing motion, construct the objective function which gives the measure of the quality of a given motion, and build a constraint function that ensures implementability. Both of these functions have to produce numeric

978-1-4244-4195-2/09/$25.00 (c) 2009 IEEE

output for any inputs in a given range. Now let us see the main factors that influence a throwing motion. To produce an implementable throwing motion the following conditions have to be fulfilled:

- The end-effector (EE) acceleration must change its sign to release the thrown object.
- The cartesian position where the release occurs must be a valid configuration for the robot.
- At this point the EE velocity has to be achievable within joint velocity limitations.
- All the joints have to be able to accelerate to the release velocity within joint position limits.
- Full deceleration must be achievable within the same joint limits.
- The predefined maximal current consumption, joint velocity, acceleration and jerk limitations must be fulfilled during the whole movement.
- Furthermore, the robot has to avoid collision with itself or with other parts of the environment.

During the motion planning, the following simplifications and restrictions are performed:

- We assume that after the release time instant the contact between the body and the robot breaks and no further interaction occurs (sticking friction is neglected).
- The air drag is neglected, so the free body motion simplifies to a parabolic function of time.
- The mass of the thrown body is neglected.
- We assume that the robot has active gripping mechanism thus the orientation of the EE during the motion does not influence implementability.
- The robot motion is searched as a set of two-segment cubic splines, each describes a joint's motion from an initial zero velocity to another motionless state.

Finally the control algorithm, that is capable to drive both position and speed precisely, has to actuate the robot through the obtained joint space trajectory. During the last control part, all the disturbances from real word have to be compensated, especially but not limited to the full electric and dynamic behavior of the actuators, and the robot dynamics including mechanical self oscillations.

III. System description

The robotic arm contains six electrical motors (called PowerCubes) mounted on each other in a chain like topology. These are modular building blocks shipped by SCHUNK Intec Inc. and have good dynamical properties. The layout of the kinematic structure is similar to a published configuration [8] which was designed for ball catching purposes using same type but stronger motors. The robot is placed on the top of a self-designed support made from steel plates and a tube to maximize work space area and rigidity. Most of the coupling elements are also self-designed polyamide type plastic pipes to have a lightweight structure. The whole construction and its 3D structural model used for collision detection can be seen in figure 1. Table I lists the modified Denavit-Hartenberg (MDH) parameters according to [9] and joint angle limits for the built topology. The joint limits are due to the wiring along the robot and they help to avoid collisions between the neighboring links.

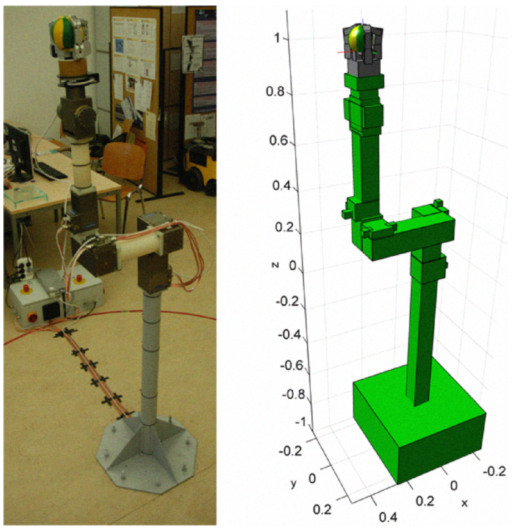

Fig. 1. The robot arm and its collision detecting model.

TABLE I
Modified Denavit-Hartenberg notation and rotational limits

i	α_{i-1}	a_{i-1}	d_i	$Pc_{i,min}$	$Pc_{i,max}$
1	0°	0	0.27 m	−180°	180°
2	−90°	0.29 m	0	−240°	60°
3	180°	0	0	−230°	50°
4	−90°	0	0.5 m	−180°	180°
5	−90°	0	0	−115°	115°
6	90°	0	0.217 m	−180°	180°

Table II lists the dynamical physical limits, namely the maximum velocity, acceleration, current consumption and output torque.

TABLE II
Actuator physical limits

motors	PR-090	PR-070	PW-070$_1$	PW-070$_2$
velocity ($°/sec$)	149	149	248	320
acceleration ($°/sec^2$)	596	596	992	1280
current ($Amp@24V$)	30	15	15	15
output torque (Nm)	206	73	54	28

Each module contains harmonic drive gearing with $1:161$ reduction ratio and a built-in electronic PID controller. From control point of view, each joint can be actuated and measured as a discrete-time system through one 1 Mbit/sec CAN bus. The PowerCubes, like many other industrial type modular robotic components, use local PID controllers for position or velocity control. However, there are no built in functions to produce controlled torque or even acceleration. With the provided C++ dynamic link library (DLL) the discrete-time acceleration commands were generated using the so called ramp

velocity profile motion which is a sequence of acceleration, constant speed and deceleration phases. The parameters (i.e., maximum acceleration and velocity during the ramp motion) can be set for each motion command as upper limits. The motors can be actuated with voltage commands as well and thanks to the built-in 2000 slots/rotation incremental encoder the repetition/measurement precision is about $0.02°$. Also the velocities can be read from the PowerCubes but their numerical computation from the position data gives the same degree of precision. The whole robot is powered by two DC 24V 30A power-supplies. The modules' hardware version is 4614 for PowerCubes PR-090 and PR-070, and 3519 for PW-070.

IV. Description of the Algorithm

Initially, the following data is given:

1) task specific information
   - $\mathbf{p}_{tr}$, the target position in world frame
2) predefined robot specific constraint
   - $\mathbf{Pc}_{min}$ and $\mathbf{Pc}_{max}$, the joint position limits
   - $\mathbf{Vc}$, the joint velocity limits
   - $\mathbf{Ac}$, the joint acceleration limits
   - $\mathbf{Jc}$, the joint jerk limitations
   - $\mathbf{C}_{max}$, the maximum allowed current consumption of each module
   - $\mathbf{Cc}$, maximum output currents of the power supplies
   - $\mathbf{T}_{ball}$, a 4x4 homogeneous transformation defining the position and release capable orientation of the load with respect to the EE frame (EEF)
   - robot kinematic (mDH parameters), dynamic and collision avoidance model
3) optimization related variables and functions
   - $\mathbf{h} = [\mathbf{p}_0, \gamma, \omega, F_l, H_s, d_r]$, optimization parameter vector
   - $\mathbf{p}_0$ is the release position in world frame given by spherical coordinates $[\theta, \phi, r]$
   - γ and ω defines orientation and angular velocity, respectively measured around the z axis of $\mathbf{T}_{ball}$ at the release time instance (referenced as the release frame (RF) with respect to the world frame)
   - F_l is the flight time of the thrown object from $\mathbf{p}_0$ to $\mathbf{p}_{tr}$
   - H_s is the duration of the whole movement
   - d_r is the ratio of the deceleration motion duration with respect to H_s
   - $\mathbf{h}_{min}$ and $\mathbf{h}_{max}$ lower and upper bounds of the search space, respectively
   - $J(\mathbf{h}) : \mathbb{R}^8 \rightarrow \mathbb{R}$, objective function
   - $U(\mathbf{h}) : \mathbb{R}^8 \rightarrow \mathbb{R}^{6N+n}$, is the constraint function, where N is the number of actuators while n is the number of power supplies

The algorithm works as follows. In each iteration for the given h input the solutions of the joint motions are searched in the form of a two-segment cubic splines. This is done in the following steps:

1) The cartesian position of $\mathbf{p}_{tr}$, the duration of the accelerating motion phase $h_a = (1 - d_r)H_s$ and the duration of the deceleration time $h_b = d_r H_s$ are computed.
2) Based on $\mathbf{p}_{tr}$, $\mathbf{p}_0$ and F_l, the initial velocity vector for the body is computed as $\mathbf{v}_0 = \frac{\mathbf{g}}{2}F_l + \frac{\mathbf{p}_{tr} - \mathbf{p}_0}{F_l}$,] where $\mathbf{g}$ is the gravity vector.
3) The computation of the vector of joint angles ($\mathbf{q}_r$) are performed according to the inverse kinematics relationship, using $\mathbf{p}_0$, the orientation of $\mathbf{v}_0$, γ and ${\mathbf{T}_{ball}}^{-1}$. From the eight symbolically obtained different solutions, the valid configurations are chosen within the two bounds $\mathbf{Pc}_{min}$ and $\mathbf{Pc}_{max}$. If no valid solution exists both $J(\mathbf{h})$ and $U(\mathbf{h})$ returns high predefined values. If multiple valid solutions are found, the one which consumes less current is chosen using the current consumption model described later.
4) To compute the joint velocities ($\dot{\mathbf{q}}_r$) for a particular $\mathbf{q}_r$ the inverse jacobian of the robot, $\mathbf{p}_0$, $\mathbf{v}_0$, ${\mathbf{T}_{ball}}^{-1}$ and ω are used.
5) The solution for the joint motions is searched in the form of cubic splines. Let $\mathbf{Q}_a(t) : \mathbb{R} \rightarrow \mathbb{R}^6$, $\mathbf{Q}_a(t) = \mathbf{a}_a t^3 + \mathbf{b}_a t^2 + \mathbf{c}_a t + \mathbf{d}_a$ denote such a multidimensional cubic polynomial (over the interval $[0,\ h_a]$) that starts from a moveless state and have the desired position and velocity at time h_a, and similarly $\mathbf{Q}_b(t) : \mathbb{R} \rightarrow \mathbb{R}^6$, $\mathbf{Q}_b(t) = \mathbf{a}_b t^3 + \mathbf{b}_b t^2 + \mathbf{c}_b t + \mathbf{d}_b$ defined on the interval $[h_a,\ H_s]$ that gives the same position and first two derivatives as $\mathbf{Q}_a$ at time h_a and zero velocity at time H_s. So the two cubic polynomials are found by solving the following system of equations:

$$\begin{aligned} \mathbf{Q}_a(h_a) &= \mathbf{q}_r = \mathbf{Q}_b(h_a), \\ \dot{\mathbf{Q}}_a(h_a) &= \dot{\mathbf{q}}_r = \dot{\mathbf{Q}}_b(h_a), \\ \dot{\mathbf{Q}}_a(0) &= \mathbf{0} = \dot{\mathbf{Q}}_b(h_a + h_b), \\ \ddot{\mathbf{Q}}_a(h_a) &= \mathbf{0} = \ddot{\mathbf{Q}}_b(h_a). \end{aligned}$$

The solution is parameterized by h_a, h_b, $\mathbf{q}_r$ and $\dot{\mathbf{q}}_r$ and given in the form below:

$$\begin{aligned} \mathbf{a}_a &= -\frac{\dot{\mathbf{q}}_r}{3h_a^2}, & \mathbf{a}_b &= -\frac{\dot{\mathbf{q}}_r}{3h_b^2} \\ \mathbf{b}_a &= \frac{\dot{\mathbf{q}}_r}{h_a}, & \mathbf{b}_b &= \frac{\dot{\mathbf{q}}_r h_a}{h_b^2} \\ \mathbf{c}_a &= \mathbf{0}, & \mathbf{c}_b &= \dot{\mathbf{q}}_r - \frac{\dot{\mathbf{q}}_r h_a^2}{h_b^2} \\ \mathbf{d}_a &= \mathbf{q}_r - \dot{\mathbf{q}}_r \frac{2h_a}{3}, & \mathbf{d}_b &= \mathbf{q}_r + \dot{\mathbf{q}}_r(-h_a + \frac{h_a^3}{3h_b^2}). \end{aligned}$$

The optimization can be started from different initial conditions chosen randomly along all dimensions of $\mathbf{h}$ between $\mathbf{h}_{min}$ and $\mathbf{h}_{max}$ using a general constrained nonlinear optimization method such as the `fmincon` function in the Matlab Optimization Toolbox.

The collision avoidance is solved using a model which covers the rigid volume of the arm with oriented bounding boxes (see fig. 1). The base and each link contains 3-5

boxes whose position and orientation are updated through forward kinematics. None of these boxes allowed to have any overlapping and this property is checked by separating axis theorem tests [10]. Assuming that no joint limit violation is present, the number of box collision checks can be reduced considerably. This checking takes place for finely discretized time instants of the optimized motion with zero tolerance (i.e. no collision should occur at the examined time instants). This computation have to be done separately because it produces a binary (yes/no type) result for the possible occurrence of collision.

A. Objective function

The objective function computes the weighted sum of terms that characterize the behavior of the throwing motion, namely

$$J(\mathbf{h}) = k_T H_s + k_B b_r + k_J \sum_{j=1}^{N} \int_0^{t_f} (\dddot{q}_j)^2 \, dt \qquad (1)$$

where k_T, k_B, and k_J, are positive weighting parameters. k_T, and k_J correspond to time and jerk minimal movements, respectively. Parameter k_B helps to achieve slowly accelerating but fast decelerating motion to ensure that that the robot really loses contact with the thrown object after the time instance h_a.

It is possible that for a particular value of h the objective function cannot be evaluated because no configuration is capable to reach the given release position and orientation. In this case, a high objective function value is returned. It must be admitted that this solution may degrade optimization performance. We note however, that there exist some optimization methods, that are able to handle unsuccessful objective function evaluations.

B. Constraints

To find only implementable solutions all the given constraints have to be fulfilled, namely

$$\begin{aligned}
q_j(t) &\leq Pc_{j_{max}}, \quad j = 1, \ldots, N && (2)\\
q_j(t) &\geq Pc_{j_{min}}, \quad j = 1, \ldots, N && (3)\\
|\dot{q}_j(t)| &\leq Vc_j, \quad j = 1, \ldots, N && (4)\\
|\ddot{q}_j(t)| &\leq Ac_j, \quad j = 1, \ldots, N && (5)\\
|\dddot{q}_j(t)| &\leq Jc_j, \quad j = 1, \ldots, N, && (6)
\end{aligned}$$

where N is the number of actuators, $Pc_{j_{min}}$ and $Pc_{j_{max}}$ indicates minimum and maximum joint positions, furthermore Vc_j, Ac_j and Jc_j are given velocity, acceleration and jerk bounds, respectively for each joint j. Other necessary but more difficult constraints are applied, regarding the power consumption of the robot arm, namely

$$\begin{aligned}
\max_{t\in[0,t_t+t_b]} I_j(t) &\leq C_{max\,j}, \quad j = 1, \ldots, N && (7)\\
\max_{t\in[0,t_t+t_b]} \sum_{j\in \mathbf{list}_k} I_j(t) &\leq Cc_k, \quad k = 1, 2, && (8)
\end{aligned}$$

where $\mathbf{list}_k$ lists the index of motors connected to the kth power supply, and $I_j(t)$ is the current consumed by the jth motor at time t, more precisely

$$I_j(t) = K_{Bj}|\dot{q}_j(t)| + K_{Mj}|\tau_j(\mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t))|, \qquad (9)$$

where K_{Bj} represents constant slope between joint velocity and current consumption of motor j, K_{Mj} is the slope of the linear correspondence between armature current and output torque for the jth motor and $\tau_j(\mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t))$ is the computed torque acting on joint j in the actual robot state, regarding desired position, velocity and acceleration of the joints.

The solutions for $\mathbf{Q}_a(t)$ and $\mathbf{Q}_b(t)$ are monotone functions, so the constraints (2)-(3) can be checked by computing only $\mathbf{Q}_a(0)$ and $\mathbf{Q}_b(h_a + h_b)$. The left hand sides of inequalities (4) will reach their maximum at h_a, thus it is only necessary to compute either $\dot{\mathbf{Q}}_a(h_a)$ or $\dot{\mathbf{Q}}_b(h_a)$. For constraints (5), the only $\ddot{\mathbf{Q}}_a(0)$ and $\ddot{\mathbf{Q}}_b(h_a + h_b)$ have to be considered, while the computation of (4) simplifies to choosing the maximum values from $6|\mathbf{a}_a|$ and $6|\mathbf{a}_b|$. For the current consumption limits we used the simplifying assumption that motor dynamics is dominant over load torques, which means that currents mainly depend on motor velocities. Thus only $\tau(\mathbf{Q}_a(h_a), \dot{\mathbf{Q}}_a(h_a), \mathbf{0})$ have to be computed to determine the $I_j(h_a)$'s.

C. Initial conditions and parameters

In general optimization problems, the initial condition has a great role both in the overall running time and the convergence properties. Thus, some kind of exploration was used to store information about those evaluation attempts that did not produce valid solution. Every time the constraint function is called, if the inverse kinematic solution exists, the computed output is checked which inequalities are completely fulfilled from (2)-(8). Then the parameter vector is stored in a database according to which of the inequalities are satisfied. When the optimization is started next time, the initial condition can be set close to those values that previously produced (at least partially) valid solution. Practically, the most complete solution is preferred, and this iteration is done till the optimization finds an optimal solution.

V. The Controller

Based on the fact that the actuating motors have enough power and they have built-in PID controllers, a decentralized higher level control scheme was used based on the individual trajectory tracking control of the joints. Let us denote the position and velocity of joint i by $z_{1,i}$ and $z_{2,i}$, respectively. Furthermore, let us assume that the manipulable input variable is the joint acceleration command (v_i). If we denote the reference for $z_{1,i}$ and $z_{2,i}$ by $z_{1ref,i}$ and $z_{2ref,i}$, respectively, then the tracking error system can be written as

$$\dot{e}_{1,i} = e_{2,i}, \quad \dot{e}_{2,i} = u_i \qquad (10)$$

where $e_{1,i} = z_{1,i} - z_{1ref,i}$, $e_{2,i} = z_{2,i} - z_{2ref,i}$ and $u_i = v_i - \dot{z}_{2ref,i}$. Thus, by the asymptotic stabilization of the dynamics (10) the asymptotic tracking of the reference trajectory is assured.

There are known physical constraints for the acceleration inputs v_i that must be satisfied during operation. The minimum-maximum values for v in the case of the different joints can be given in the following vector form

$$v_{min/max} = -/+[10.4\ 10.4\ 10.4\ 10.4\ 16.6\ 22.3] rad/sec^2 \tag{11}$$

Furthermore, we know the minimum and maximum acceleration values of the designed joint trajectories and denote them by a_{min} and a_{max}, respectively. From this, it is clear that the physical constraints (11) will be fulfilled, if the following inequalities hold for the transformed input u in (10):

$$v_{min,i} - a_{min,i} \leq u_i \leq v_{max,i} - a_{max,i}, \text{ for } i = 1, \ldots, 6 \tag{12}$$

The following well-known result from linear control theory is used for the design of a state feedback with input constraints [11].

Theorem 1. Consider the linear system $\dot{x} = Ax + Bu$, where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^p$, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times p}$. There exists a stabilizing state feedback K such that for every initial condition $x(0)$ with $\|x(0)\| \leq 1$, the resulting control input $u = Kx$ satisfies $\|u(t)\| \leq u_{max}$ for $t \geq 0$ if and only if there exists a positive definite symmetric $n \times n$ matrix Q and a $p \times n$ matrix Y such that the following linear matrix inequality (LMI) constraints are satisfied:

$$\begin{aligned} &-AQ - QA^T - BY - Y^T B^T > 0 \\ &\begin{bmatrix} u_{max}^2 I & Y \\ Y^T & Q \end{bmatrix} \geq 0. \end{aligned} \tag{13}$$

If the LMIs (13) are feasible, then the feedback gain can be computed as $K = YQ^{-1}$. (We note that '> 0' ad '≥ 0' in (13) denote positive definiteness and semidefiniteness, respectively.)

To apply *Theorem 1*, we have to symmetrize the input constraints (12). This does not introduce significant conservatism in this case, since the obtained minimum and maximum bounds for the control inputs are almost symmetric to zero. Furthermore, a linear state transformation is applied that transforms the set of possible initial conditions into the unit circle.

The controller implementation was programmed in a sequential manner using time stamping for all incoming and outgoing data. In the presence of six motors on the CAN bus, the time between two forthcoming extended ramp motion command to the same motor was approximately 17 ms. The PowerCube's acceleration reference is given as the maximum acceleration value parameter of the mentioned *PCube_moveRampExtended* command. In this case, the module internal circuitry tries to reach the given maximal acceleration and this limit can be updated with every extended motion command. Only a suitable target position have to be given beside the desired acceleration value to guide the motors in the appropriate direction. For example, this value can be the module's actual position ± 3 rad which guarantee that the started ramp motion acceleration phase is not finished until the next command cycle ends on the CAN bus.

VI. Simulation and Measurement Results

A. Optimization results

In figure 2, valid and optimized trajectories can be seen for the target position $\mathbf{P}_{tr} = [1.0, 1.0, -1.0]$ m. On the bottom of the figure, the computed overall current consumption is drawn. The dashed lines show the current consumption for the two power supplies. The optimization was done with parameter values $k_T = 15$, $k_B = 20$, and $k_J = 0.01$, furthermore the search space was limited by

$$\mathbf{h}_{min} = [-\frac{\pi}{2}, -\frac{\pi}{2.5}, 0.5, -\pi, -\frac{\pi}{2}, 0.3, 0.2, 0.04]$$
$$\mathbf{h}_{max} = [\frac{\pi}{2}, \frac{\pi}{2}, 1.0, \pi, \frac{\pi}{2}, 3.0, 2.0, 0.4],$$

and the optimized output was

$$\mathbf{h} = [0.493, -0.465, 0.858, 0.583, -1.098, 0.586, 0.920, 0.391].$$

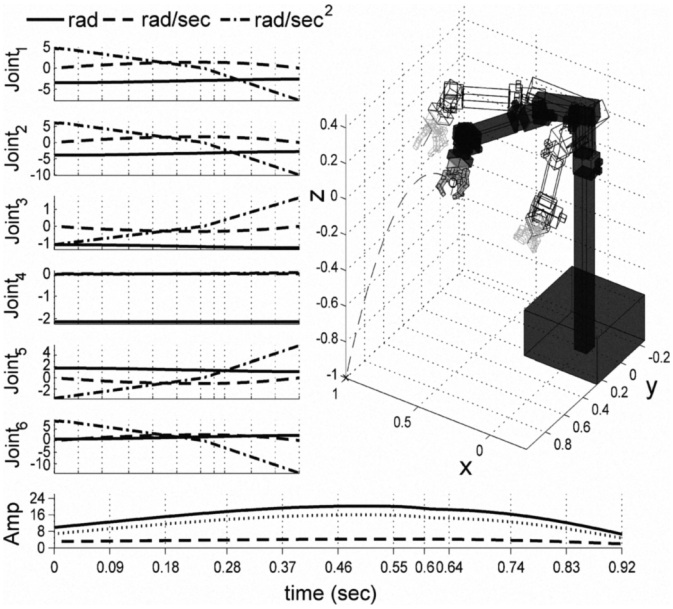

Fig. 2. Optimized trajectory within maximum current constraints.

During the development of the algorithm it turned out, that even for a short range throwing motion the DC motors consume a lot of current due to the high joint velocities at release time. It also became clear, that even more than current limits, the joint position limitations and the collision avoidance restriction badly reduced the set of feasible solutions and necessitate the collection of partially feasible solutions.

B. Joint control measurements

The robot arm was driven through an optimized throwing trajectory to check tracking error and overall current consumption in practice. The optimization parameters were $k_V = 0.1$, $k_F = -10$, $k_A = -10$, $k_D = 20$ and $k_O = 1$. Figure 3 shows position (line) and velocity (dashed line) tracking errors for each joint, which after an initial 0.1 second transient stays under 0.05 rad and 0.1 rad/sec, respectively. The time scale

and time lags can be shown on the bottom of the picture. The measured current consumption is plotted on the upper axis. The measured (continuous line) values follow the predicted consumption (dashed line) but the unmodeled effects like the frictional terms and mechanical resonances cause a variating shift between measured and computed currents.

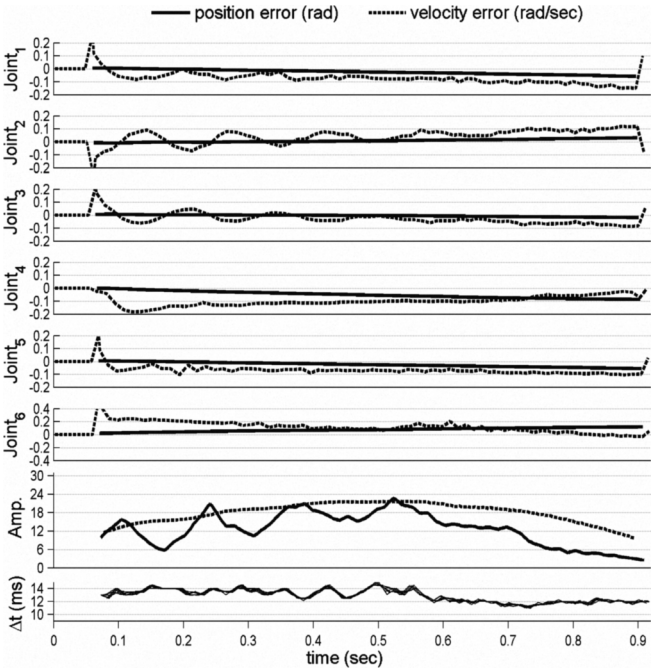

Fig. 3. **a)** Reference tracking measurements for joint 2, **b)** Phases of the implemented throwing motion

C. Motor parameter estimation

To compute current consumption for each motor j, two parameters had to be determined, namely K_{Mj} and K_{Bj}. K_{Bj} was measured as the average current consumption of the jth motor at stabilized maximal velocities for constant voltage input without any inertial load. Than the the slope of the linear relationship was determined by regression. Values for K_{Mj} were determined by measurement pairs of computed loads and currents that compensated them. Then obtaining linear regressors between joint torques and applied currents represented by slopes stored as K_{Mj}'s. The resulted parameters are listed in Table III, signed (*) values are given by the manufacturer.

TABLE III
ACTUATOR PARAMETERS

$motor_i$	$K_{Bj}(secAmp/rad)$	$K_{Mj}(Amp/Nm)$
$1,2,3$	2.9116	0.1419
4	0.7347	0.0567
5	0.4871	0.013*
6	0.1572	0.025*

VII. Conclusions and Future Work

The throwing motion generation of a 6-DOF rigid robot arm was shown in this paper. The trajectory tracking of the joints is performed using a discrete-time linear controller. The proposed solution is implemented on a real system, and the early measurements show good results. The final aim is the real-time use of the the proposed motion generation method, but this may necessitate the modification of some components of the algorithm and/or the change of the computation hardware platform possibly to a processor array (the different versions of which are widely available today). The proposed linear approximation for current consumption estimation gave usable results, but additional work to polish the current model is needed. Currently, the implemented optimization can be used to obtain preprocessed motion tasks for higher level control methods.

VIII. Acknowledgements

This research was partially supported by the grant no. OTKA K67625. The second author is a grantee of the Bolyai János Research Scholarship of the Hungarian Academy of Sciences. Special thanks are given to Prof. Tamás Roska for his great support and guidance during the project. The authors also thank for all the help and review supported by the members of the robotic laboratory at the Faculty of Information Technology of Péter Pázmány Catholic University.

References

[1] Werner Heisenberg. *Physics and beyond; encounters and conversations*, volume 42 of *World perspectives*. Harper & Row, New York, NY, USA, 1971. Translated from the German by Arnold J. Pomerans.

[2] N. Kato, K. Matsuda, and T. Nakamura. Adaptive control for a throwing motion of a 2 DOF robot. In *Proc. of the 4th International Workshop on Advanced Motion Control, AMC '96-MIE*, volume 1, pages 203–207, 1996.

[3] K.M. Lynch and M.T Mason. Dynamic nonprehensile manipulation: controllability, planning and experiments. *International Journal of Robotics Research*, 18 (1):1999, 1998.

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

[5] J. E. Bobrow. Optimal robot plant planning using the minimum-time criterion. *IEEE Journal of [see also IEEE Transactions on Robotics and Automation] Robotics and Automation*, 4(4):443–450, August 1988.

[6] L. C. T. Wang and C. C. Chen. A combined optimization method for solving the inverse kinematicsproblems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):489–499, August 1991.

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

[8] C Smith and H.I. Christensen. Using cots to construct a high performance robot arm. In *2007 IEEE International Conference on Robotics and Automation*, pages 4056–4063, Rome, Italy, 2007.

[9] John J. Craig. *Introduction to robotics: mechanhics and control*. Pearson Prentice Hall, Berlin, Germany, third edition, 2005.

[10] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.

[11] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in Systems and Control Theory*. SIAM Books, 1994.