

Trajectory tracking control of a 6-degree-of-freedom robot arm using nonlinear optimization

Ferenc Lombai¹

¹Faculty 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}

²Systems 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

The trajectory tracking control of a 6-degree-of-freedom (DOF) rigid robot arm is described in this paper. The trajectories for the joint variables are generated in third-order spline form using general constrained nonlinear optimization, taking into consideration the joint position, velocity, acceleration, jerk and overall current consumption constraints during the movement. The trajectory tracking of the individual joints is solved using a discrete-time linear controller design. The obtained trajectories are previously checked to avoid collisions using oriented bounding boxes and their separating axis theorem tests. The complete inverse kinematics of the arm is symbolically calculated in the Mathematica computing environment and implemented in C++. Simulations and measurements show the applicability of the proposed method.

I. INTRODUCTION

The modelling and control of mechanical structures and robot manipulators in particular has been an intensively studied area with numerous application fields [1], [2]. For many practical robotic problems, the motion planning task can be handled in the framework of a general constrained nonlinear optimization problem that is nontrivial to parametrize and solve [3].

The solution of the above mentioned problems are now supported with effective software tools such as the Matlab and Mathematica environments for numeric and symbolic computations. Other more specific examples are the RobotBuilder application [4] that can be used to rapidly develop robotic dynamic simulations and the ROBOOP object oriented C++ package [5].

In this paper, we present an extended implementation of the algorithm [6] for planning time-jerk optimal trajectories on a 6-DOF robot arm. From practical reasons, the original method has been extended by involving into the nonlinear optimization additional constraints related to joint variable limits and to the instantaneous power consumption of the actuating motors.

The structure of the paper is the following. Section II contains the brief description of the experimental system, section III describes the extended trajectory planning algorithm while the trajectory tracking control scheme can be found in section IV. The most important simulation and measurement results are shown in section V, and section VI concludes the paper with a summary.

II. SYSTEM DESCRIPTION

A. Physical system structure

The robotic arm contains six electrical motors mounted to each other in a chain like topology. These motors called PowerCubes are shipped by Amtec-Robotics Inc. and have great dynamical properties and robust configuration possibilities like water resistant coverage. These modular building blocks can be used to achieve a wide range of kinematic configurations with appropriate coupling elements. The layout of the kinematic structure is the first main issue to solve for a desired manipulation task. To achieve fast trajectory tracking, a published configuration [7] is used which is designed for ball catching purposes using same type but stronger motors. The robot is mounted on the top of a self-designed support made from steel plates and 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 assembly and its model used for collision detection can be seen in figure 1.

Table I lists the modified Denavit-Hartenberg (MDH) parameters according to [8] and joint angle limits for the built topology. The joint limits are due to wiring along the robot and lets avoid collision between the neighboring links.

Besides structural constraints, the used motors have dynamic limitations, too. Table II lists these physical limits, namely the maximum velocity, acceleration, current consumption and output torque.

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 overheating reported in [7] was not experienced, because the average read/access cycle period in each module is only about 5 ms due to the shared bus. The PowerCubes, like many other industrial type modular robotic components, use local

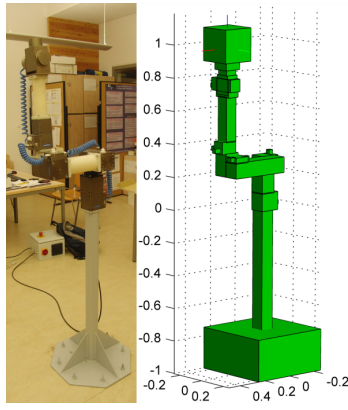


Fig. 1. The robot arm and its collision detecting model.

TABLE I
MODIFIED DENAVIT-HARTENBERG NOTATION AND ROTATIONAL LIMITS

i	α_{i-1}	a_{i-1}	d_i	$q_{i_{min}}$	$q_{i_{max}}$
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
ACTUATOR PHYSICAL LIMITS

motors	PR-090	PR-070	PW-070 ₁	PW-070 ₂
velocity ($^\circ/sec$)	149	149	248	320
acceleration ($^\circ/sec^2$)	596	596	992	1280
current (Amp@24V)	30	15	15	15
output torque (Nm)	206	73	54	28

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 current commands, too 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 gave the same degree of precision. The whole robot is powered with a DC 24V 30A power-supply. The modules' hardware version is 4614 for PowerCubes PR-090 and PR-070, and 3519 for PW-070.

B. Inverse kinematics

The first necessary and nontrivial problem for every manipulator is the inverse relationship between the end effector frame (EEF) and the joint angles, namely

$$\mathbf{q} = K^{-1}(\mathbf{T}_{EEF}^0),$$

where \mathbf{q} denotes the vector of the joint variables and \mathbf{T}_{EEF}^0 is a homogeneous transformation matrix that defines the position and orientation of the EEF with respect to the world frame. This notation stands for the problem to solve a redundant set of complicated trigonometric equations. For general cases with $\text{DOF} > 6$, no closed form solutions are guaranteed and even for 6 DOF case the computation for different sets of nonzero MDH parameters can be complicated [9] [10]. Fortunately, the proposed kinematic chain is very similar to the well studied Puma 560's structure and the last three rotational joints also compose a spherical wrist. Therefore eight different closed form solutions can be given for a desired EEF from the 16 general possible solutions. The inverse kinematic algorithm was constructed in the Mathematica computing environment, translated to C++ and then included into the Roboop object oriented framework. This module allows the user to give task specific targets

in the work space. The algorithm gives a maximum of eight solutions that fit into the joint limits. Average running time is measured to be 0.4 ms on Windows XP operating system with a Core Duo 3.6 GHz CPU. For comparison, the accessible algorithm on the Popeye robot website [7] under same conditions but in MATLAB environment executed in about 0.2 ms.

C. Dynamical model

The dynamical equations were symbolically derived with the help of [11] and have the following common form:

$$\mathbf{T} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) \quad (1)$$

Where \mathbf{q} , $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ are joint angles, velocities, and accelerations, respectively \mathbf{M} is the symmetric joint-space inertia matrix, or manipulator inertia tensor, \mathbf{C} represents Coriolis and centripetal effects, \mathbf{F} is the term for viscous and Coulomb friction (often neglected in practice), \mathbf{G} is the gravitational load and \mathbf{T} is the vector of torques applied at joints. It is straightforward to express $\ddot{\mathbf{q}}$ from (1) as follows:

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q}) (\mathbf{T} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{F}(\dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q})) \quad (2)$$

To carry out the numeric computation of the torques, a MATLAB SimMechanics model have also been used neglecting the frictional terms. The inertia properties of the links were computed in the RobotBuilder software environment, assuming uniform density distribution inside the modules, neglecting screws, spacers and wires. Tables III and IV list the main dynamic parameters used for computations.

TABLE III
LINK MASS(kg) AND CENTER OF MASS (cm)

Link _i	mass	cg _{x_i}	cg _{y_i}	cg _{z_i}
1	3.3	-0.174	0	-6.75
2	3.45	14.5	-0.167	9
3	2.55	0.088	4.424	0
4	1.45	0	-0.224	21.43
5	1.2	0	0	0
6	0.3	0	0	-14.5

TABLE IV
LINK INERTIA PROPERTIES (cm²kg)

Link _i	I _{xx_i}	I _{xy_i}	I _{xz_i}	I _{yy_i}	I _{yz_i}	I _{zz_i}
1	0.678036	0	0	0.678036	0	1.01554
2	2.53819	0	0	9.6059	0	9.6059
3	4.18382	0.018877	0	1.27925	0	4.18382
4	44.9309	0	0	45.3309	-11.5519	44.4205
5	23.225	0	0	14.9	0	18.125
6	1.267	0	0	1.267	0	1.734

III. TRAJECTORY (PATH) PLANNING

The design and control environment into which the trajectory planning algorithm is embedded, uses the off-line concepts from [12] with the following structure. The task planning algorithm based on the task and environment specifications (for example, pick and place target positions) composes the movements in Cartesian space as a rare sequence of target frames and feeds them to the path planning section. These frames then translated to joint space vectors. The inverse kinematic progress produces up to eight different solutions. So as the simplest case, a sequence of joint space via-points is selected based on the initial configuration by choosing the closest forthcoming (and valid) solution. It is assumed, that the given target points in the joint space are checked to fit into the given joint limits and are free of collision. 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 updated through forward kinematics. None of these boxes allowed to have any overlapping and this property is checked by separating axis test theorem tests [13]. Assuming that no joint limit violation present, the number of box collision checks can be reduced considerably. Then the trajectory planning method computes the necessary execution time for each translation between the target points so that the first two derivatives of the joints' motion become continuous functions (appropriate polynomials). This path is passed to the trajectory generator which computes the joint space curve considering the hardware limitations. The controller grabs this information and drives the manipulator through

the given state space trajectory using a feedback described later. During the last control part, all the disturbances from real world have to be compensated.

The trajectory planning method was largely based on [6], where the following problem is solved in the space of joint variables. Minimize the objective function

$$J(h) = k_T N \sum_{i=1}^{v_p-1} h_i + k_J \sum_{j=1}^N \int_0^{t_f} (\ddot{q}_j)^2 dt \quad (3)$$

subject to the constraints

$$\begin{aligned} |\dot{q}_j(t)| &\leq VC_j, \quad j = 1, \dots, N \\ |\ddot{q}_j(t)| &\leq WC_j, \quad j = 1, \dots, N \\ |\dddot{q}_j(t)| &\leq JC_j, \quad j = 1, \dots, N, \end{aligned} \quad (4)$$

where N is the number of robot joints, q_j is the position of the j th joint, v_p is the number of via-points in the joint variable space, $h = [h_1 \ h_2 \ \dots \ h_{v_p-1}]^T$ is the optimized variable denoting the time-intervals between two via-points, $t_f = \sum_{i=1}^{v_p-1} h_i$ is the total execution time of the trajectory, k_T and k_J are positive weighting parameters. Furthermore, VC_j , WC_j and JC_j are given velocity, acceleration and jerk bounds, respectively for the j th joint. The solution is in the form of cubic splines and let $Q_{j,i}(t)$ denote such a polynomial for the j th joint defined on the interval h_i , i.e. on $[t_i, t_{i+1}]$. After setting the initial conditions appropriately, the minimization problem (3)-(4) can be solved using a general constrained nonlinear optimization method such as the `fmincon` function in Matlab.

The implemented optimization minimizes the weighted sum of the total execution time and the squared two-norm of the jerk. With the mentioned continuity criteria, the intermediate accelerations can be calculated by solving a set of linear equations. The original algorithm in [6] checks maximal jerk, acceleration and velocity constraints but it does not check position limits. Therefore, to avoid position constraint violation, the extreme values of positions are also computed inside the constraint function by solving the equation

$$\begin{aligned} \dot{Q}_{i,j}(t) &= \frac{h_i(\alpha_i - \alpha_{i+1})}{6} + \frac{q_{i+1,j} - q_{i,j}}{h_i} + \\ &+ \frac{\alpha_{i+1,j}(t - t_i)^2 - \alpha_i(t_{i+1} - t)^2}{2h_i} = 0 \end{aligned}$$

for t . If no real solution exists, then the spline is a monotonous function over the interval. If a solution lies inside the time interval $[t_i, t_{i+1}]$ the position should be checked against the joint's limits. So the constraints are supplemented with the following ones:

$$\begin{aligned} q_j(t) &\leq PC_j^{max}, \quad j = 1, \dots, N \\ q_j(t) &\geq PC_j^{min}, \quad j = 1, \dots, N \end{aligned}$$

where PC_j^{min} and PC_j^{max} indicates minimum and maximum joint positions for each joint j .

Another necessary but difficult constraint is the overall power consumption of the robot arm. To remain inside the power-supply limit, the actuator dynamics and robot dynamics have to be considered together. The aim is to find the instance during motion, when the consumption is maximal. To obtain an easily computable power consumption model, we use the simplifying assumption that motor dynamics is dominant over load torques, which means that currents mainly depend on motor velocities. We also neglect the motor shaft inertia, inductivity term and mechanics related damping. Therefore it is good approximation to find a time instance inside the i th interval, when the weighted sum of the joint velocities reaches its maximum, namely

$$\max_{t \in [t_i, t_{i+1}]} \sum_{j=1}^N (K_{Bj} |\dot{q}_j(t)|), \quad (5)$$

where K_{Bj} parameter represents the linear relationship between constant current and maximal unloaded rotational velocity of the motors, and can be easily determined for each joint j . Because the velocities are quadratic functions, they have extrema either on the interval ends or at an intermediate point. Due to the absolute values in (5), the searched intermediate point can be found from maximum of $4N + 3$ possibilities, namely the two interval ends t_i and t_{i+1} , $2N$ points where $\dot{q}_j(t) = 0$ thus the derivative of the sum in (5) is not defined and $2N + 1$ point between the previous ones, where the signed and weighted sum of the accelerations cross zero. Let's denote the time instance by t_i^* where the weighted sum of velocities is found to be maximal. To consider inertial loads, the inverse dynamics is computed for each t_i^* regarding desired position, velocity and acceleration of the joints. The computed torque loads are also added to the previously computed candidate current consumption maximum

multiplied by the linear relationship K_{M_j} between output torque and armature current based on the DC motor and propulsion model described in [2]. Then the estimated maximal current I_m for the overall motion is computed by the expression

$$I_m = \max_{i=1, \dots, v_p-1} \sum_{j=1}^N (I_{gap_j} + K_{B_j} |\dot{q}_j(t_i^*)| + K_{M_j} T_j(t_i^*)),$$

where I_{gap_j} represents maximal current that still not moves the unloaded motor j , K_{M_j} is the slope of the linear correspondence between armature current and output torque for the j th motor and $T_j(t_i^*)$ is the computed torque acting on joint j in the desired robot state at time t_i^* .

An other modification of the original algorithm is also applied to increase optimization convergence robustness. For each interval length lower and upper bounds were set to 0.001 and 20 seconds, respectively.

IV. TRAJECTORY TRACKING CONTROL OF THE ACTUATORS

Based on the already mentioned facts that the actuator motors have a sufficiently high power and they have built-in PID controllers (that cannot be turned off), a decentralized control scheme was used for the trajectory tracking control of the individual joints that is based on the pole-placement design for linear systems. Let us denote the joint variable of a certain joint by z_1 and the joint velocity by z_2 . Furthermore, let us assume that we can directly prescribe the joint acceleration by an appropriate control input (u). Then the linear state space model for the dynamics of a joint reads

$$\dot{z}_1 = z_2 \quad (6)$$

$$\dot{z}_2 = u \quad (7)$$

Let us denote the required reference trajectory for the joint variable by z_{1ref} . Then the reference for z_2 is $z_{2ref} = \dot{z}_{1ref}$. The tracking errors are given by

$$e_1 = z_1 - z_{1ref}, \quad e_2 = z_2 - z_{2ref} \quad (8)$$

The time-derivatives of the tracking errors are the following

$$\dot{e}_1 = e_2 \quad (9)$$

$$\dot{e}_2 = v \quad (9)$$

where $v = u - \dot{z}_{2ref} = u - \ddot{z}_{1ref}$. Thus, by asymptotically stabilizing the error dynamics (9), the asymptotic tracking of the reference trajectory is guaranteed. For this, a simple pole-placement method was used to prescribe the eigenvalues of the error dynamics [14]. The computed feedback is a linear full state feedback of the form

$$v(t) = -K_e(t) = -k_1 e_1(t) - k_2 e_2(t) \quad (10)$$

where the feedback gains k_1 and k_2 can be computed using the `place` command of Matlab's Control System Toolbox.

The implementation of the controller is in discrete time, where v is computed as

$$v(k) = -K_d e(k) = -k_{d1} e_1(k) - k_{d2} e_2(k) \quad (11)$$

i.e., the actual control input is given by

$$u(k) = -k_{d1} e_1(k) - k_{d2} e_2(k) + \dot{z}_{1ref}(k) \quad (12)$$

The discrete time controller gain K_d is computed in the following standard way. The closed loop continuous error dynamics (9)-(10) is converted to discrete time using a selected sampling rate assuming zero order hold on the input. Then the poles p_{d1} , p_{d2} of the controlled discrete-time system are computed, and finally, a pole placement design in discrete time is performed to find a feedback K_d such that the poles of the closed-loop discrete time error dynamics are p_{d1} and p_{d2} .

The controller implementation was programmed in a sequential manner using time stamping for all incoming and outgoing data. The controller error feedback gain vector K_d was computed previously according to the average interaction interval for the discrete-time system. 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 16.4 ms. The computed discrete-time feedback gain was $K_d = [69.149 \ 16.153]$ with the closed loop poles $p_{d1} = 0.877$ and $p_{d2} = 0.8487$. 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 $\pm(\pi - 0.1)$ which guarantee that the started ramp motion acceleration phase is not finished until the next command cycle ends on the CAN bus.

V. SIMULATION AND MEASUREMENT RESULTS

Firstly, the identification of the actuator parameters will be shown followed by the results of the trajectory planning algorithm. Then the operation of the trajectory tracking control will be presented by simulation and measurement results.

A. Motor parameter estimation

To compute current consumption for each motor j , three parameters had to be determined, namely I_{gap_j} , K_{M_j} and K_{B_j} . I_{gap_j} and K_{B_j} were determined by measuring stabilized maximal velocities for constant current input to the unloaded motors. Then the linear relationship was determined by the Matlab `robustfit` regression command. I_{gap_j} is the shift and K_{B_j} is the slope of the defined correspondence. Values for K_{M_j} were determined by measurement pairs of computed loads and measured currents that compensated them. Then the resulted values were reduced by the corresponding I_{gap_j} rates thus obtaining linear regressors between joint torques and applied currents represented by slopes stored as K_{M_j} 's. The resulted parameters are listed in Table V, signed(*) values are only approximations based on K_B values.

TABLE V
ACTUATOR PARAMETERS

$motor_i$	$I_{gap_j}(Amp.)$	$K_{B_j}(secAmp/rad)$	$K_{M_j}(Amp/Nm)$
1, 2, 3	2.7754	0.0464	10.2280
4	1.2514	0.0255	12.2736*
5	0.4212	0.0154	16.3649*
6	0.8626	0.0070	14.3193*

B. Optimization results

To show the effects of the constraints on the optimization result, target positions were chosen close to the joint limits. In figure 2, the optimized trajectories can be seen. On the bottom of the figure, the computed overall current consumption is drawn. The dash-dot line shows overall current based on velocity and I_{gap} , while the dashed line represents calculated current containing computed torque terms. The target intervals are plotted by triangles and in each interval h_i at time instances t_i^* the values of the two curves are highlighted by squares and circles, respectively. The optimization was done with $k_T = 10$ and $k_J = 1$ values.

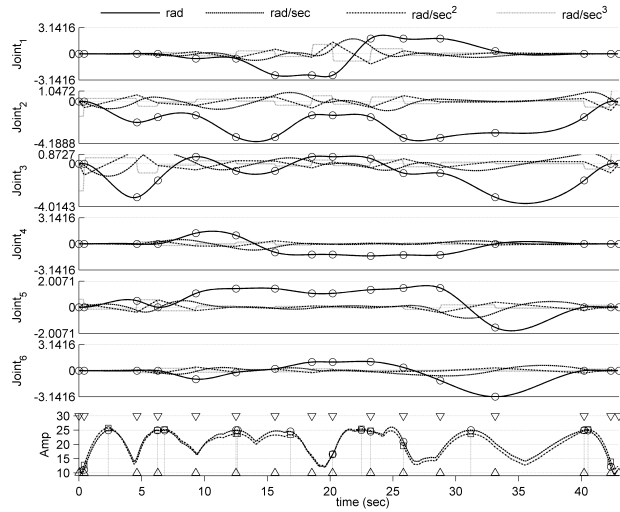


Fig. 2. Optimized trajectory, with maximum current constraint.

C. Joint control simulations

In figure 3, a Simulink simulation of the designed controller can be seen driving the second joint over the optimized trajectory. The joint's initial condition was intentionally shifted by -90° to visualize vanishing tracking error over the first 5 seconds. It can be seen that no position overshoot occurs during the stabilization transient. It is also visible that regarding the reference velocity, some overshoot is produced temporarily increasing the motor currents. This can be avoided for a planned trajectory if the initial conditions are appropriately matched.

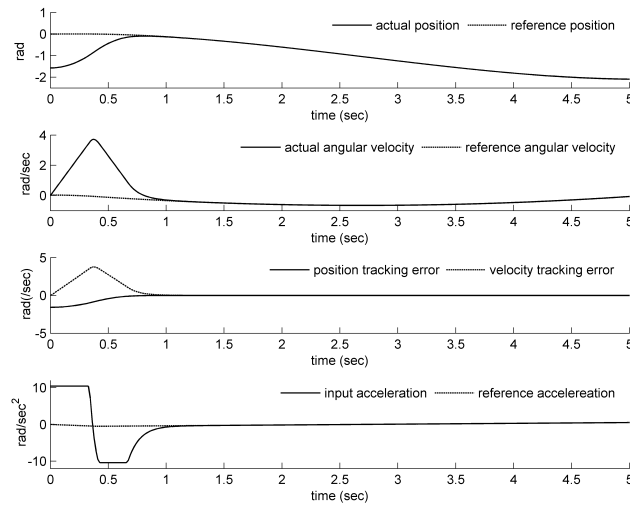


Fig. 3. The discrete time pole placement joint control simulation results.

D. Trajectory tracking measurements

Finally, the robot arm was driven through the optimized trajectory to check tracking error and overall current consumption in practice. Figure 4 shows position (line) and velocity (dashed line) tracking errors for each joint, which after an initial 0.2 second transient stays under 0.02 rad and 0.03 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 but the unmodelled effects and the three approximately modelled motors (see table V) cause a varying shift between measured and computed currents. The maximum positive difference was in the range of 5 A, thus the optimization constraint was set to 25 A which proved to be good practical upper bound for generating motion that is always inside the current limits.

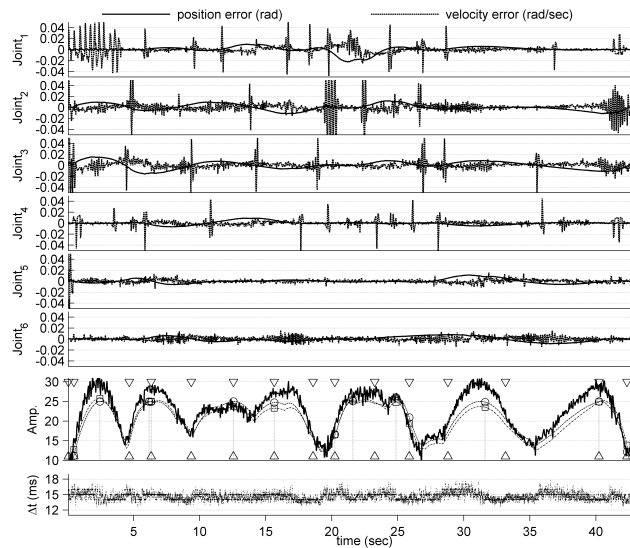


Fig. 4. Tracking error and current consumption measurements.

The biggest flaw of the generated motion is a small stop near zero velocity instances. This can be caused by mechanical damping like friction in the propulsion chain or the energy step needed to change the direction of magnetic flux in the armature coils. Moreover, there are other mechanical or software design properties which cause minimum possible velocity and acceleration inputs in the range of $1.7 \cdot 10^{-5} \text{ rad/sec}$ and 0.04 rad/sec^2 , respectively. Figure 5 shows the reference accelerations and controller computed input accelerations for each joint during the optimized trajectory tracking.

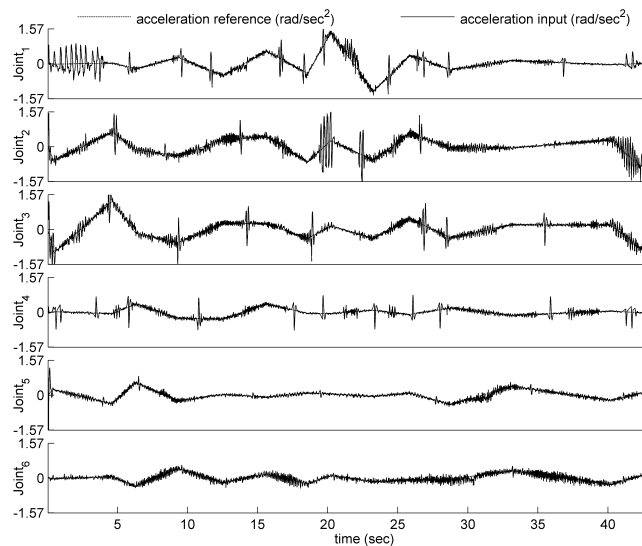


Fig. 5. Acceleration reference and control input.

VI. CONCLUSIONS AND FUTURE WORK

The trajectory tracking control of a 6-DOF rigid robot arm was shown in this paper. The planned via-points in the operational space are transformed into the joint space using the inverse kinematic relations computed in Mathematica. The original trajectory planning algorithm [6] has been extended to take into consideration joint and current limits. The trajectory tracking of the joints is performed using a discrete-time linear controller. The proposed solution has been implemented and tested on a real system. Although there are great simplifications in some of the models, the measurements show good performance. Almost each of the control blocks can be further improved so that the overall system gives a basic framework to guide a robot arm through a desired set of Cartesian positions with orientation safely. The proposed linear approximation gave usable results, but additional work on the current consumption model is needed. The implemented optimization can be used to obtain preprocessed motion tasks for higher level control methods. Furthermore, communication lags during data transfer could be reduced using a more advanced hardware/software environment.

ACKNOWLEDGEMENTS

This research was partially supported by the grant no. OTKA F046223. 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] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators, 2nd Edition*. Springer-Verlag, 2000.
- [2] Mark W. Spong. *Robot Dynamics and Control*. John Wiley & Sons, Inc., New York, NY, USA, 1989.
- [3] S.M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
- [4] Steven Rodenbaugh and David E. Orin. Robotbuilder. <http://www.ece.osu.edu/orin/robotbuilder/robotbuilder.html>.
- [5] R. Gourdeau. *ROBOOP. A Robotics Object Oriented Package in C++*. Documentation. Ecole Polytechnique de Montreal, 2007.
- [6] A. Gasparetto and V. Zanutto. A technique for time-jerk optimal planning of robot trajectories. *Robotics and Computer-Integrated Manufacturing*, in press:doi:10.1016/j.rcim.2007.04.001, 2007.
- [7] 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.
- [8] John J. Craig. *Introduction to robotics: mechanics and control*. Pearson Prentice Hall, Berlin, Germany, third edition, 2005.
- [9] Martin Pfurner Manfred L. Husty and Hans-Peter Schröcker. A new and efficient algorithm for the inverse kinematics of a general serial 6r manipulator. *Mechanism and Machine Theory*, 42:66–81, 2007.
- [10] L. Yang, H. Fu, and Z. Zeng. A practical symbolic algorithm for the inverse kinematics of 6r manipulators with simple geometry. *Lecture Notes in Computer Science*, 1249:73–86, 1997.
- [11] P.I. Corke. An automated symbolic and numeric procedure for manipulator rigid-body dynamic significance analysis and simplification. In *1996 IEEE International Conference on Robotics and Automation*, pages 1018–1023, Minneapolis, USA, 1996.
- [12] Akihiko Matsushita and Takeshi Tsuchiya. Control system design with on-line planning for a desired signal and its application to robot manipulators. 3(2):149–152, June 1998.
- [13] 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.
- [14] K.J. Astrom and B. Wittenmark. *Computer Controlled Systems: Theory and Design*. Prentice Hall, 1984.