Real-Time Trajectory Generation Using Model Predictive Control

Ghazaei, Mahdi; Olofsson, Björn; Robertsson, Anders; Johansson, Rolf

Published in:
2015 IEEE Conference onAutomation Science and Engineering (CASE)

2015

Real-Time Trajectory Generation using Model Predictive Control*

M. Mahdi Ghazaei Ardakani[1], Björn Olofsson[1], Anders Robertsson[1], and Rolf Johansson[1]

***Abstract*— The problem of planning a trajectory for robots starting in an initial state and reaching a final state in a desired interval of time is tackled. We consider Model Predictive Control as an approach to the problem of point-to-point trajectory generation. We use the developed strategy to generate trajectories for transferring the state of the robot, fulfilling computational real-time requirements. Experiments on an industrial robot in a ball-catching scenario show the effectiveness of the approach.**

I. Introduction

Trajectory generation is an inherent problem in motion control for robotic systems, such as industrial manipulators and mobile platforms. The motion-planning problem is to define the path, and the course of motion as a function of time, namely the trajectory. In many applications, the trajectory generation is desired to be performed such that the time for executing a task, or the energy consumed during the motion is minimized. Hence, motion-planning problems are often formulated as optimal control problems [20]. For solving a trajectory-planning problem, it is usually beneficial to solve the path-planning problem first and then find a trajectory by assigning time to each point along the path [32]. Nevertheless, for the specification of the problem, such separation is unnecessary in the optimization framework. Therefore, as long as there is no computational constraint, both path planning and trajectory planning can be solved in an integrated approach. Ultimately, this leads to a motion planning where the full potential of the system is utilized.

In the motion-planning procedure, various modeling assumptions have to be made. The major difference with regard to this is if a kinematic or a dynamic model of the system is considered. Although dynamic models are required for achieving the highest possible performance, a kinematic model suffices in many applications, given conservative constraints on the kinematic variables—*i.e.*, on the position, the velocity, and the acceleration [10].

A desired characteristic of the motion planning in uncertain environments is the ability to react to sensor inputs [17]. In this paper, we study a scenario with a ball-catching robot [21], where the estimates of the contact position of the ball are delivered online from a vision system. As more vision data become available, the position can be estimated with lower uncertainty. Thus, a new trajectory needs to be computed when new estimates are delivered. Our approach to trajectory generation is based on Model Predictive Control (MPC) [26], [24] and the receding-horizon principle. By employing a final-state constraint in the MPC formulation, it is possible to solve fixed-time motion-planning problems where analytic solutions are not available. In contrast to the typical application of MPC for reference-tracking problems, we adopt a trajectory-generation perspective. In our experiments, we use the aforementioned strategy to compute reference values (joint position and velocity reference values) for an underlying motion-control system.

A. Previous Research

Methods for trajectory generation based on polynomial functions of time were already developed in the 1970s [28], [31]. Time-optimal path tracking for industrial manipulators was suggested in the 1980s [5], [30]. An overview of trajectory-generation methods for robots is provided in [17]. There are many methods for online trajectory generation with the objective of time-optimality based on analytic expressions, parametrized in the initial and final states of the desired motion and certain constraints [7], [23], [12], [18]. The difference between these methods is in their generality with respect to the initial and final states and constraints.

The major advantage of the existing analytic solutions is that nearly time-optimal or time-optimal trajectories can be computed extremely fast. While the minimum-time trajectories are of interest for defining an upper bound for productivity of a robotic system, they put the system under maximal stress. Hence, the solution to fixed-time problems with a cost function motivated by the application may prove valuable by reducing the wear of the robot. A sub-optimal solution to the fixed-time trajectory planning for robot manipulators was proposed in [9]. In another approach, the fixed-time optimal solution in the space of parametrized trajectories was found [33].

Trajectory generation based on optimization using a kinematic model has been proposed in [3] for ball-catching robotic systems. Methods for catching flying objects with robots were also considered in [22], [19]. In the latter, a dynamic model was employed and machine-learning algorithms were used in the online execution. MPC has been proposed for control and path tracking for mobile robots; see, *e.g.*, [14], [15], [16]. In [29], MPC is used for tracking of linearly interpolated way-points. A two-layer architecture is proposed in [27] with a generic optimal-control formulation at the high-level and MPC at the low-level for tracking the solution generated by the high-level planner.

*The research leading to these results has received funding from the European Union's seventh framework program (FP7/2007-2013) under grant agreements PRACE (Ref. #285380) and SMErobotics (Ref. #287787). The authors are members of the LCCC Linnaeus Center and the ELLIIT Excellence Center at Lund University.

[1]The authors are with the Department of Automatic Control, LTH, Lund University, SE–221 00 Lund, Sweden. The corresponding author is M. M. Ghazaei A. E–mail: mahdi.ghazaei@control.lth.se.

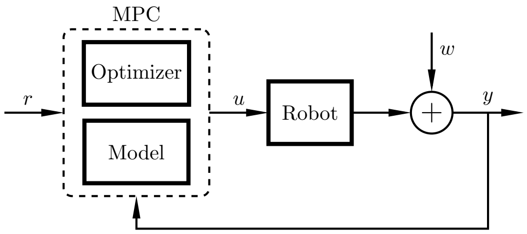

Fig. 1. Schematic description of trajectory generation using MPC; r is the reference signal, u is the control signal, y is the output signal, and w is the output disturbance. The feedback from the output to the MPC is not present if an open-loop strategy is used.

B. Contributions

The major contribution of this paper is a method for fast online motion planning, given an initial state of a robot and a desired final state at a given time. The approach does not rely on any predefined trajectory or path. To this purpose, we have applied the MPC framework to point-to-point trajectory generation. Compared to previous research, in our approach it is possible to solve the trajectory-planning problem online with application-specific cost functions and a broader class of constraints on inputs and states. The algorithm relies on convex optimization, such that real-time computational constraints for modern robotic systems can be satisfied. Further contributions are a complete implementation of this approach to motion planning and an experimental evaluation in a challenging scenario of ball catching.

II. Problem Formulation

In many robotic applications, it is desirable to obtain a certain state of the robot at a given time. This will result in a reference-tracking problem if the desired state is specified as a function of time during the whole execution. On the other hand, if the desired state is unspecified during certain intervals in time, we need to do planning between the points, *i.e.*, transferring the robot from an initial state to the next state in the given time. In either case, the motion of the robot must fulfill predefined constraints. Moreover, we wish to specify a desired behavior, which is implicitly characterized by the optimum of an objective function.

We propose to use the model predictive approach to solve this type of problems. A central notion in MPC is using a model to predict the behavior of a system. Accordingly, it is possible to optimize the objective function over a receding horizon, considering the predicted states and outputs. The optimization is usually carried out at each sample. Then, only the first sample in the computed control inputs over the prediction horizon is applied [24], [26]. Figure 1 shows a schematic of trajectory planning using MPC.

In the MPC framework, it is required to define a model of the system, an objective function, state and input constraints, and a prediction horizon. We consider linear models as follows

$$x(k+1) = A_d x(k) + B_d u(k), \tag{1}$$

$$y(k) = C_d x(k) + D_d u(k), \tag{2}$$

$$z(k) = \tilde{C} x(k) + \tilde{D} u(k). \tag{3}$$

Here, $u(k)$, $x(k)$, $y(k)$, and $z(k)$ denote the control signal, the states, the measured outputs and the controlled variables, respectively [24]. We consider a quadratic cost at time k as

$$V_k(\mathcal{U}) = \sum_{i=k+1}^{k+N} \|z(i) - r(i)\|^2_{Q(i)} + \sum_{i=k}^{k+N-1} \|u(i)\|^2_{R(i)}, \tag{4}$$

where $\mathcal{U}^T = \left[u^T(k), u^T(k+1), \cdots, u^T(k+N-1)\right]$, N is the prediction horizon, the norm is defined $\|a\|^2_W = a^T W a$ for a positive semidefinite weighting matrix W, and the reference signal is denoted by $r(i)$. Additionally, Q and R are time-dependent weight matrices. Linear constraints on the control and the states are assumed as follows

$$F\mathcal{U}(k) \le f, \tag{5}$$

$$G\mathcal{Z}(k) \le g, \tag{6}$$

where $\mathcal{Z}^T = \left[z^T(k+1), z^T(k+2), \cdots, z^T(k+N)\right]$, F and G are matrices whose dimensions are determined by the considered system dynamics (1)–(3), the number of constraints, and the prediction horizon.

Given the initial state $x(k)$ of the system, the following optimization problem is solved at each sample k [26]:

$$\begin{aligned} \underset{\mathcal{U}}{\text{minimize}} \quad & V_k(\mathcal{U}) \\ \text{subject to} \quad & (1),\ (3),\ (5)\text{–}(6) \\ & z(k+N) = r_f(k), \quad \text{(optional)} \end{aligned} \tag{7}$$

where $r_f(k)$ is the constraint on the controlled variable at the final time in the prediction horizon.

It is desirable to investigate how the MPC framework is applicable to trajectory generation for point-to-point problems. Further, we wish to find a set of assumptions and methods that allow for real-time solutions.

III. Methods

In this section, we specialize the general linear MPC framework in Sec. II to point-to-point trajectory generation. We also introduce a kinematic model and a set of constraints for our experiments.

A. Point-to-Point Planning

In the general objective function defined in (4), the desired controlled states do not need to be specified at every sample. This allows point-to-point trajectory generation. To make this clear, assume that $\Psi(k) \subset \{k+1, \cdots, k+N\}$ denotes the set of indices where there are desired values for the controlled states z. By assuming that $r(i) = 0$ when $i \notin \Psi(k)$, we can rewrite (4) as

$$\begin{aligned} V_k(\mathcal{U}) = & \sum_{i \in \Psi(k)} \|z(i) - r(i)\|^2_{Q(i)} + \sum_{i=k+1}^{k+N} \|z(i)\|^2_{\bar{Q}(i)} \\ & + \sum_{i=k}^{k+N-1} \|u(i)\|^2_{R(i)}, \end{aligned} \tag{8}$$

where $\bar{Q}(i) = Q(i)$ if $i \notin \Psi(k)$ and $\bar{Q}(i) = 0$ if $i \in \Psi(k)$. For reference-tracking problems, usually the first and the last terms are important. Provided that explicit constraints for the desired controlled states are defined, the first term can be ignored in the point-to-point trajectory generation.

Nevertheless, if soft constraints are preferred, then all of the terms might be used. In this case, the benefit is immediate if the optimization problem with explicit constraints on the controlled variables $z(i)$, $i \in \Psi(k)$, is infeasible.

Since the optimization in general runs at every sampling instant, it is possible to account for the updates in the target or deviation of the robot. If good tracking for the robot is assumed, the loop can be closed around the model too, providing an open-loop control strategy (*cf.* Fig. 1).

Although it is possible to introduce constraints on the states at any sampling instant, we limit ourselves to constraints on $z(k+N)$. This strategy is advantageous if no information about the variations in the target is available ahead in time, or if we do not want this information to impact the current decision. In such cases, we can successively reduce the sampling period while keeping the number of discretization points constant—*i.e.*, the prediction horizon in discrete time remains constant while the prediction horizon in continuous time gradually decreases. This allows for improving the resolution of the solution as the system follows the computed trajectory toward the target state.

B. Interpolation of Trajectories

Because of the real-time execution of the motion planning, it is required to have a valid trajectory in every moment. This is obviously the case even if the optimization algorithm fails, for example as a result of an infeasible problem specification. Thus, we consider piece-wise trajectories. The previous trajectory is valid until the new one is ready to be switched to. The robot remains still in its final position if no other piece of trajectory for the current time exists.

Since MPC is formulated for discrete-time systems, we need to interpolate between the computed trajectory points. This makes it possible to have a lower sampling rate for the optimization than for the controlled system. Assuming that T is a vector of increasing time instants and that $\mathcal{X}$ is a vector with the corresponding states,

$$T = \begin{bmatrix} t_1, & t_2, & \cdots, & t_n \end{bmatrix},$$
$$\mathcal{X} = \begin{bmatrix} x_1, & x_2, & \cdots, & x_n \end{bmatrix},$$

we consider a linear interpolation such that

$$x(t) = x_k + \frac{x_{k+1} - x_k}{t_{k+1} - t_k}(t - t_k), \qquad t_k \le t < t_{k+1}. \tag{9}$$

C. Discretization

In the MPC framework, the dynamics of a system is described by difference equations. Assuming the following continuous-time linear model

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{10}$$
$$y(t) = Cx(t), \tag{11}$$

the discrete-time system obtained using the predictive first-order-hold sampling method [2] with a sampling period h is

$$\begin{aligned} x(k+1) = \Phi x(k) + \frac{1}{h}\Gamma_1 u(k+1) \\ + (\Gamma - \frac{1}{h}\Gamma_1)u(k), \end{aligned} \tag{12}$$
$$y(k) = Cx(k), \tag{13}$$

where

$$\begin{bmatrix} \Phi & \Gamma & \Gamma_1 \end{bmatrix} = \begin{bmatrix} I & 0 & 0 \end{bmatrix} \exp\left(\begin{bmatrix} A & B & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} h \right). \tag{14}$$

Note that (12) can be rewritten in the standard form by changing to a new state variable ζ according to

$$\zeta(k+1) = \Phi\zeta(k) + \left(\Gamma + \frac{1}{h}(\Phi - I)\Gamma_1\right)u(k), \tag{15}$$
$$y(k) = C\zeta(k) + \frac{1}{h}C\Gamma_1 u(k). \tag{16}$$

The choice of discretization method is motivated by the linear interpolation of the trajectories as defined in (9).

D. Model and Constraints

To illustrate the approach to trajectory generation, we consider a robot model using only the kinematic variables. Specifically, we consider multiple decoupled chain of integrators as our model. Since robots have various structures, we choose the generalized coordinates q for the representation of the states. In this way, q may represent either the joint values or the Cartesian coordinates depending on the application. We choose the state vector as

$$x = \begin{bmatrix} q_1, \dot{q}_1, \ddot{q}_1, \cdots, q_d, \dot{q}_d, \ddot{q}_d \end{bmatrix}^T,$$

where d is the number of Degrees of Freedom (DoF).

The continuous-time model (10)–(11), where each DoF is assumed to be a triple integrator, is specified by the matrices

$$A = \text{blkdiag}\left([\tilde{A}, \cdots, \tilde{A}]\right), \; B = \text{blkdiag}\left([\tilde{B}, \cdots, \tilde{B}]\right),$$

where blkdiag$(\cdot)$ forms a block diagonal matrix from the given list of matrices, $A \in \mathbb{R}^{3d\times 3d}$, $B \in \mathbb{R}^{3d\times d}$, and

$$\tilde{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \tilde{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Assuming $\tilde{C} = I_{3d}$, the matrices for the discretized model concerning the controlled states can be calculated as below

$$\Phi = \text{blkdiag}\left([\tilde{\Phi}, \cdots, \tilde{\Phi}]\right), \; \Gamma_1 = \text{blkdiag}\left([\tilde{\Gamma}_1, \cdots, \tilde{\Gamma}_1]\right),$$
$$\Gamma = \text{blkdiag}\left([\tilde{\Gamma}, \cdots, \tilde{\Gamma}]\right),$$

where $\Phi \in \mathbb{R}^{3d\times 3d}$, $\Gamma_1, \Gamma \in \mathbb{R}^{3d\times d}$, and

$$\tilde{\Phi} = \begin{bmatrix} 1 & h & \frac{h^2}{2} \\ 0 & 1 & h \\ 0 & 0 & 1 \end{bmatrix}, \; \frac{1}{h}\tilde{\Gamma}_1 = \begin{bmatrix} \frac{h^3}{24} \\ \frac{h^2}{6} \\ \frac{h}{2} \end{bmatrix}, \; \tilde{\Gamma} = \begin{bmatrix} \frac{h^3}{6} \\ \frac{h^2}{2} \\ h \end{bmatrix}.$$

It is straightforward to include constraints on linear combinations of the kinematic variables. From a practical perspective, we consider limiting the coordinate positions, the velocities, the accelerations, and the jerks. We form matrices F, G and vectors f, g in (5)–(6) such that

$$|u(i)| \le \begin{bmatrix} u^1_{\max}, & \cdots, & u^d_{\max} \end{bmatrix}^T, \qquad k \le i \le k+N-1$$
$$z^j_{\max} = \begin{bmatrix} q^j_{\max}, & v^j_{\max}, & a^j_{\max} \end{bmatrix}, \qquad 1 \le j \le d$$
$$|z(i)| \le \begin{bmatrix} z^1_{\max}, & \cdots, & z^d_{\max} \end{bmatrix}^T, \qquad k+1 \le i \le k+N$$
$$z(k+N) = r_f(k),$$

where $r_f(k)$ is the desired final state according to (7).

By assuming the objective function (8) with $\Psi(k) = \emptyset$ and $\bar{Q}(k+N) = 0$, *i.e.*, ignoring the first term and the cost on the final state, we have a full specification of the MPC problem for point-to-point trajectory generation. Note that the desired target state is a constraint on the final state. This implies that we can successively reduce the sampling period, as described in the last paragraph in Sec. III-A.

IV. Implementation

The method described in Sec. III-D was implemented and experimentally evaluated on an industrial robot system. The system consisted of an IRB140 industrial manipulator [1], combined with an IRC5 control cabinet. The control system was further equipped with a research interface, ExtCtrl [4], in order to implement the proposed trajectory-generation method as an external controller. The research interface permits low-level access to the joint position and velocity controllers in the control cabinet at a sample period of 4 ms. More specifically, reference values for the joint positions and velocities can be specified and sent to the joint controllers, while the measured joint positions and velocities and the corresponding reference joint torques were sent back to the external controller from the main control cabinet with the same sample period. The implementation of the trajectory generation was made in the programming language Java and the communication with the external robot controller was handled using the Labcomm protocol [8].

The inverse kinematics of the robot was required in order to transform the desired target point in Cartesian space to joint-space coordinates that can be used in the motion planning. The required kinematics were also available in Java from a previous implementation [21]. For implementation of the solution of the MPC optimization problem, the CVXGEN code generator [25], [6] was used. The generator produced C code, which subsequently was integrated with the Java program using the Java Native interface. The generated C code was by default optimized for performance in terms of time complexity. It enabled solution of the required quadratic problem in the MPC within 0.5 ms—*i.e.*, each optimization cycle including overhead required approximately 1–4 ms (for the four DoF used in the considered task and a prediction horizon of 20 samples) on a standard personal computer with an Intel i7 processor with four cores. In order to preserve the numerical robustness of the solver also in the case of short sampling periods, *i.e.*, when the current position is close to the target, a scaling of the equations, the inputs, and the state constraints was introduced in the optimization. This is important, considering the fact that the matrices might be poorly conditioned when close to the final time.

For evaluating the performance of the trajectory-generation method in a challenging scenario, we considered the task of catching balls with the robot. We employed the computer-vision algorithms and infrastructure developed in [22], [21] for detection of the balls thrown and prediction of the target point. Two cameras detected the ball thrown toward the robot, and the image-analysis algorithm estimated the position and velocity, which provided the basis for the model-based prediction of the target state. A photo of the experimental setup is shown in Fig. 2. A movie showing the ball-catching experiment is accessible online [11].

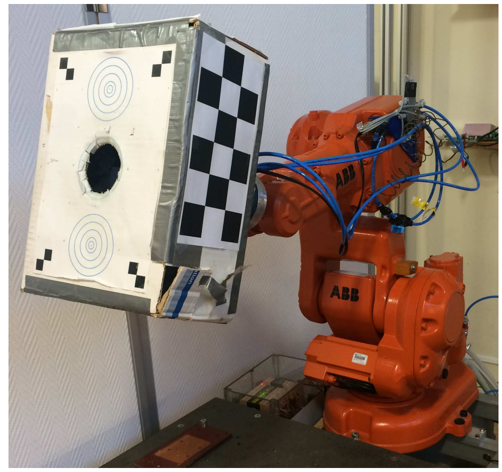

Fig. 2. The experimental setup used for evaluation. The task is to catch a ball in the box attached to the end-effector; two cameras (not visible in the figure) were used for detection of the ball.

V. Experiments and Results

The proposed approach to trajectory generation following the details in Sec. III-D was evaluated in three different experiments. The experiments concerned trajectory generation on joint level. The first experiment was performed in simulation, whereas the second and the third experiments were executed on the robot setup described in Sec. IV. In the experiments on the robot, the open-loop strategy to trajectory generation was employed—*i.e.*, no feedback from the robot measurements was used. The prediction horizon was set to 20 samples. We assumed that the DoF can be independently controlled. For each DoF, we employed the constant weighting matrices $Q = \text{blkdiag}([0,\,1,\,1])$ and $R = 0.001$. This means that we penalize a quadratic function of velocity, acceleration and jerk according to (8). In contrast to the jerk, which is the input to the adopted model, we used the computed joint velocity and position values as the reference inputs to the robot.

A. Single Target Point

In the first experiment, a single target state was provided to the trajectory generator. Starting at rest at $q = 0$ rad, the desired state to reach was $q = 1$ rad at time $t = 1$ s with $v = 0.5$ rad/s and $a = 0$ rad/s^2. The constraints were chosen as $q_{\max} = 2$ rad, $v_{\max} = 1.2$ rad/s, $a_{\max} = 100$ rad/s^2, and $u_{\max} = 250$ rad/s^3. The results of the experiment are provided in Fig. 3. Initially, a trajectory with comparably low time-resolution was computed, as an approximation of the true trajectory for transferring the system from the initial state to the final state. With a period of 200 ms, the same target state was sent to the trajectory generator. A new optimal trajectory was computed with the initial state being the state at the current time obtained from the previously computed trajectory. This resulted in a new trajectory with increased time-resolution. Considering that this procedure was repeated during the execution, a sequential refinement

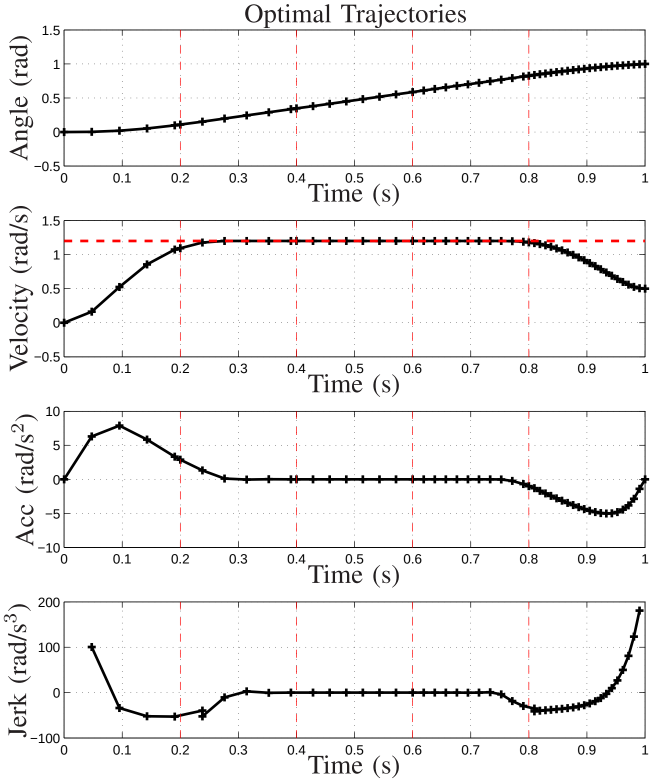

Fig. 3. Results from a simulation where the same target point was sent to the trajectory generator at the time instants indicated by the vertical, dashed red lines. A clear increase in the time-resolution of the trajectory is observed when moving closer to the target. Also, it is obvious that the constraint on the velocity is active during a major part of the motion.

of the trajectory was achieved as the system moved closer to the target state where accuracy was required, see Fig. 3.

B. Sequence of Target Points

In the second experiment, a sequence of different target states with 2 s between each new target state was provided to the trajectory generator. The different target points were located at the perimeter of a rectangle centered at the home position of the robot end-effector in the workspace. Thus, these targets required motion along different Cartesian directions of the robot. Each target state was desired to be reached after 200 ms, with zero velocity and acceleration. The constraints in the optimization were chosen based on physical properties of the joints [1] with some margin, according to $q_{\max} = 2$ rad, $v_{\max} = \pi$ rad/s, $a_{\max} = 45$ rad/s^2, and $u_{\max} = 1500$ rad/s^3 for all joints. Each target state was sent with a sample period of 20 ms, resulting in the successive refinement of the trajectory as described in the previous subsection. Once the robot reached the target state and paused there for 50 ms, a new trajectory for returning to the home position was computed and executed. Figure 4 shows the trajectories for the angle, velocity, and acceleration of joint 1 and 2.

In order to further evaluate the performance of the trajectory generator, a detailed view of the position reference given by the optimal trajectory and the corresponding measured position for joint 2 is illustrated in Fig. 5. The figure indicates the time instants at which the target state was sent and

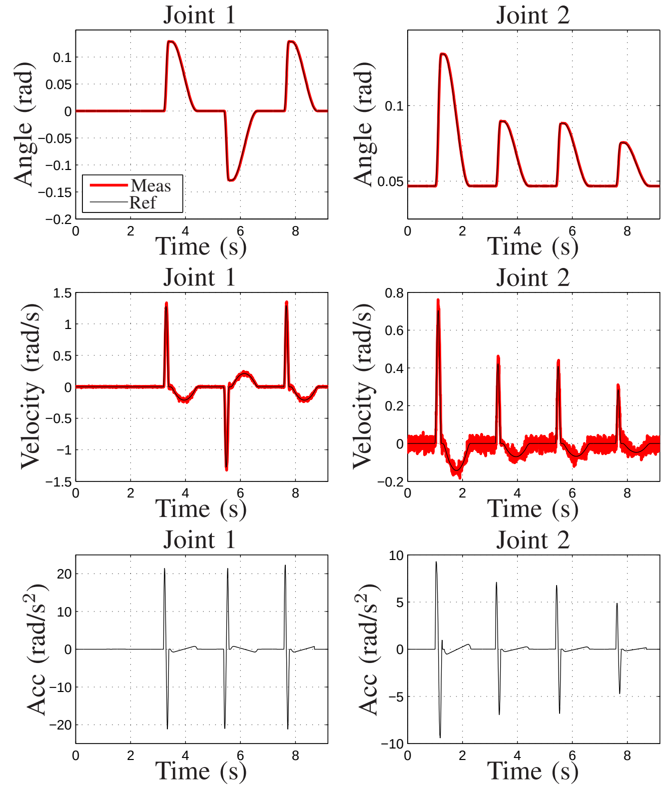

Fig. 4. Experimental results from joint 1 and 2 of the robot where a sequence of four target states was sent to the trajectory generator, each followed by a new target state coinciding with the home position of the robot. The good tracking of the computed optimal trajectories is clear from the experiments with a sequence of target states, corresponding to different points in the robot workspace.

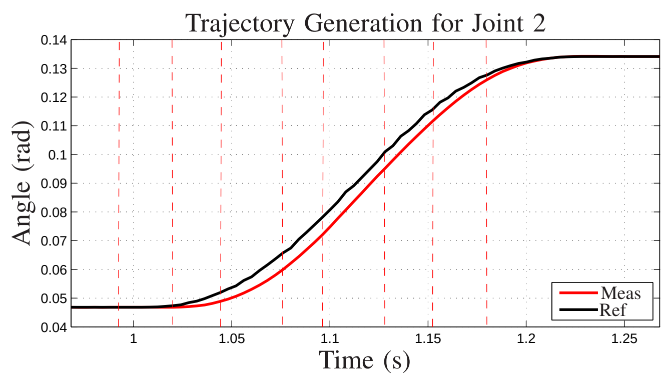

Fig. 5. A detailed view of one of the motion segments for joint 2 in Fig. 4. The time instants when the target state was sent to the trajectory generator are indicated by the vertical, dashed red lines in the figure. The delay between the reference and the actual trajectory is approximately 8 ms.

consequently a new trajectory generation was performed. The computation time for each trajectory generation was below the sample period of the robot controller, which meant that the optimal trajectory could be executed immediately. Further analysis of the computation time is presented in Sec. V-C.

C. Ball-Catching Experiments

The most demanding evaluation performed was the computations of optimal trajectories for the robot to catch balls thrown toward it. The time period from the release of the ball until it reached the robot was distributed in the interval

$[200, 800]$ ms. Hence, the success rate of the task depended on the satisfaction of the real-time constraints for computing a trajectory for transferring the robot from the home position to the desired final state at the predicted arrival time. During the flight of the balls, as soon as new data from the vision sensors and image-analysis algorithms were available, the estimated contact point (and the corresponding arrival time) of the ball was updated, leading to recomputations of the optimal trajectory. Since not all DoF of the robot were required for the task, joints 4 and 6 were not used during the experiments. The trajectories were generated such that the robot should be at the target point with zero velocity and acceleration at the predicted arrival time with a predefined margin. After reaching the final target, the robot paused there shortly in order to catch the ball, and finally returned to the home position. In the case that the estimated arrival time was already passed, no optimal trajectory was computed. If the robot, given the velocity, acceleration, and jerk constraints, could not meet the arrival time—*i.e.*, the optimization problem was infeasible—we still computed a trajectory for the robot to reach the target position at an estimate of the minimum required time. This improved the ball-catching performance since the initial target-point estimates exhibited larger uncertainty. By moving toward the target point, there was a higher chance of catching the ball later when more accurate estimates were obtained. The constraints were chosen in the same way as in Sec. V-B.

Several experiments were performed with balls thrown with random initial velocities and along different directions. The results with regard to trajectory generation from one representative experiment are presented in Fig. 6 for the joints that were active in the robot motion. It is clear that the robot tracks the position references (computed by the trajectory generator) closely. In the lower left plot in the figure, the time instants at which new sensor data arrived are also indicated. During the motion of the ball toward the robot, the computer-vision algorithm sent the current estimate of the contact point and arrival time at an approximate sampling period of 4 ms, initiating a trajectory generation with a possibly updated target state. The online replanning is also visible in the trajectory data shown in Fig. 6. The figure also shows the time margin—*i.e.*, the difference between the estimated arrival time and the earliest possible time for reaching the ball, given the constraints.

In order to verify that the real-time requirement of the trajectory-generation method was satisfied in this task, the computation times for all four DoF were measured for 100 cycles of optimization during a sequence of ball throwing. The results are visualized in Fig. 7. It can be observed that all the computation times except one are within the sampling period of the robot at 4 ms, with the average and standard deviation being 2.744 ± 0.264 ms.

VI. Discussion

As an alternative to previously suggested methods for online trajectory generation with real-time constraints for robots, we have proposed an approach based on Model Predictive Control. The main characteristic of this method is that it allows generation of trajectories that are optimal with respect to a quadratic cost function while satisfying linear constraints on the input and state variables. In contrast to earlier research, in particular [18], [12], and [23], our method gives the solution of the fixed-time problem and adds more freedom in the formulation of the motion-planning problem. Note that there is an important difference between our method and methods based on a fixed trajectory profile [28], [31], such as a fifth-order polynomial, or time scaling [13] of minimum-time solutions. In neither of these methods, it is possible to guarantee that the solution will both fulfill the state, the input, and the final-time constraints and result in the lowest possible value of the objective function.

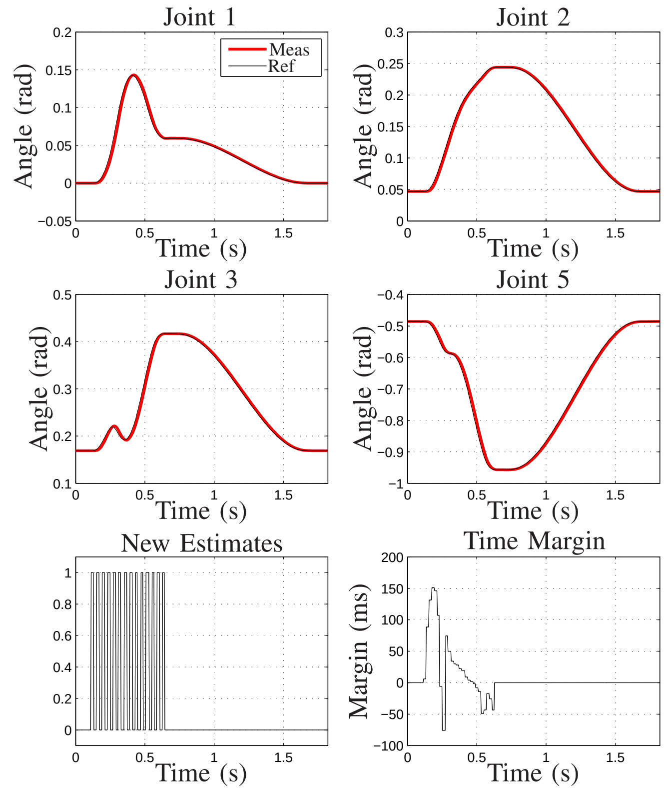

Fig. 6. Results from a ball-catching experiment. As the ball moved toward the robot, new estimates of the contact position and the arrival time were obtained and thus the trajectory was recomputed. A new estimate was obtained on the edge of the signal shown in the lower left plot. The lower right plot shows the time margin as defined in the text.

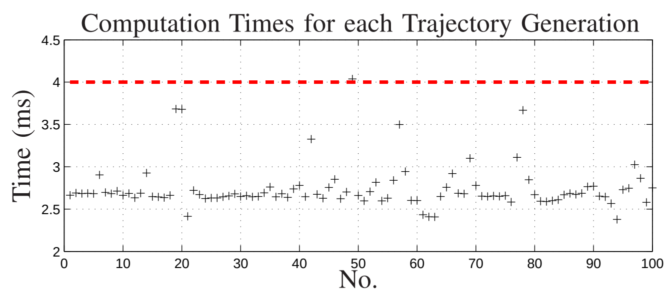

Fig. 7. The computation times for the trajectory generation during ball-catching experiments in 100 different executions (four joints were employed). The horizontal, dashed red line represents the sampling period of the robot system.

Another characteristic of our method is that it improves the accuracy of the optimal trajectory as the system approaches the desired final state, by increasing the time-resolution. Hence, despite a small number of discretization points in the initial trajectory generation, the trajectory is successively being refined.

The considered MPC solution is based on linear systems and linear constraints. This makes it possible to obtain a convex problem, which does not suffer from local optima and can be solved efficiently. Although we used a chain of integrators in Sec. III-D, more complex linear models, such as the ones modeling eigenmodes of the robot structure, could be considered.

We expect an improvement of the computation time with a more efficient implementation of the algorithms. In cases where each DoF can be independently controlled, such as for the model discussed in Sec. III-D, an efficient way to reduce the time-complexity is to distribute the computation for each DoF on a separate core of the CPU. This will allow scaling of the described algorithm to a high number of DoF, since the major part of the computation time is spent on solving convex optimization problems. Reducing the computation time allows an increased resolution of the discretization grid, if prompted by the accuracy requirements of the task.

VII. Conclusion

Model Predictive Control can offer a framework for generating trajectories, which goes beyond tracking problems. A subset of MPC problems—*e.g.*, with quadratic cost functions and linear constraints—has the potential of being efficiently implemented. This fact allowed us to make a real-time implementation of point-to-point trajectory planning for the task of ball-catching, with the feature of successive refinement of the planned trajectory. In extensive experiments, the method was evaluated in the demanding and time-critical ball-catching task with online updates of the estimated target state from a vision system as the ball approached the robot.

References

[1] ABB Robotics. ABB IRB140 Industrial Robot Data sheet, 2014. Data sheet nr. PR10031 EN_R15.

[2] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems: Theory and Design*. Prentice Hall, Englewood Cliffs, NJ, 1997.

[3] B. Bauml, T. Wimbock, and G. Hirzinger. Kinematically optimal catching a flying ball with a hand-arm-system. In *IEEE/RSJ Int. Conf. Robots and Systems (IROS)*, pages 2592–2599, Taipei, Taiwan, 2010.

[4] A. Blomdell, I. Dressler, K. Nilsson, and A. Robertsson. Flexible application development and high-performance motion control based on external sensing and reconfiguration of ABB industrial robot controllers. In *Proc. Workshop "Innovative Robot Control Architectures for Demanding (Research) Applications", IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 62–66, Anchorage, AK, 2010.

[5] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *Int. J. Robotics Research*, 4(3):3–17, 1985.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge Univ. Press, Cambridge, UK, 6th edition, 2004.

[7] R. H. Castain and R. P. Paul. An on-line dynamic trajectory generator. *Int. J. Robotics Research*, 3(1):68–72, 1984.

[8] Dept. Computer Science, Lund University. Labcomm protocol, 2015. URL: http://wiki.cs.lth.se/moin/LabComm.

[9] I. Duleba. Minimum cost, fixed time trajectory planning in robot manipulators. A suboptimal solution. *Robotica*, 15(05):555–562, 1997.

[10] M. M. Ghazaei Ardakani. Topics in trajectory generation for robots. Licentiate Thesis ISRN LUTFD2/TFRT--3265--SE, Dept. Automatic Control, Lund University, Sweden, 2015.

[11] M. M. Ghazaei Ardakani, B. Olofsson, A. Robertsson, and R. Johansson. Movie showing the ball-catching experiment. `http://www.control.lth.se/media/Research/ROBOT/PRACE/trajGenCASE15.mp4`, 2015.

[12] R. Haschke, E. Weitnauer, and H. Ritter. On-line planning of time-optimal, jerk-limited trajectories. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, pages 3248–3253, Nice, France, 2008.

[13] J. M. Hollerbach. Dynamic scaling of manipulator trajectories. *J. Dynamic Systems, Measurement, and Control*, 106(1):102–106, 1984.

[14] T. M. Howard, C. J. Green, and A. Kelly. Receding horizon model-predictive control for mobile robot navigation of intricate paths. In *Field and Service Robotics*, pages 69–78. Springer, 2010.

[15] K. Kanjanawanishkul and A. Zell. Path following for an omnidirectional mobile robot based on model predictive control. In *IEEE Int. Conf. Robotics and Automation*, pages 3341–3346, Kobe, Japan, 2009.

[16] G. Klančar and I. Škrjanc. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems*, 55(6):460–469, 2007.

[17] T. Kröger. *On-Line Trajectory Generation in Robotic Systems*, volume 58 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, Germany, 1st edition, 2010.

[18] T. Kröger and F. M. Wahl. On-line trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *IEEE Trans. Robot.*, 26(1):94–111, 2010.

[19] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters. Trajectory planning for optimal robot catching in real-time. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 3719–3726, Shanghai, China, 2011.

[20] S. M. LaValle. *Planning Algorithms*. Cambridge Univ. Press, Cambridge, UK, 2006.

[21] M. Linderoth. *On Robotic Work-Space Sensing and Control*. PhD Thesis, Dept. Automatic Control, Lund University, Sweden, 2013. ISRN LUTFD2/TFRT--1098--SE.

[22] M. Linderoth, A. Robertsson, K. Åström, and R. Johansson. Object tracking with measurements from single or multiple cameras. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 4525–4530, Anchorage, AK, 2010.

[23] S. Macfarlane and E. A. Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Trans. Rob. Autom.*, 19(1):42–52, 2003.

[24] J. M. Maciejowski. *Predictive Control with Constraints*. Addison-Wesley, Boston, MA, 1999.

[25] J. Mattingley and S. Boyd. CVXGEN: A Code Generator for Embedded Convex Optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[26] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[27] C. Norén. Path planning for autonomous heavy duty vehicles using nonlinear model predictive control. Technical report, Dept. E.E., Linköping University, Sweden, 2013. LiTH-ISY-EX–13/4707–SE.

[28] R. Paul. Manipulator Cartesian path control. *IEEE Trans. Syst., Man, Cybern.*, 9(11):702–711, 1979.

[29] D. H. Shim and S. Sastry. An evasive maneuvering algorithm for UAVs in see-and-avoid situations. In *American Control Conf. (ACC)*, pages 3886–3891, New York, NY, 2007.

[30] K. G. Shin and N. D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Trans. Autom. Control*, 30(6):531–541, 1985.

[31] R. H. Taylor. Planning and execution of straight line manipulator trajectories. *IBM J. Research and Development*, 23(4):424–436, 1979.

[32] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Trans. Autom. Control*, 54(10):2318–2327, 2009.

[33] J. Yang, M. Dong, and Y. Tang. A real-time optimized trajectory planning for a fixed wing UAV. In *Advances in Mechanical and Electronic Engineering*, pages 277–282. Springer, Berlin, Heidelberg, Germany, 2012.