# Trajectory planning for optimal robot catching in real-time

Roberto Lampariello, Duy Nguyen-Tuong, Claudio Castellini, Gerd Hirzinger and Jan Peters

***Abstract*— Many real-world tasks require fast planning of highly dynamic movements for their execution in real-time. The success often hinges on quickly finding one of the few plans that can achieve the task at all. A further challenge is to quickly find a plan which optimizes a desired cost. In this paper, we will discuss this problem in the context of catching small flying targets efficiently. This can be formulated as a non-linear optimization problem where the desired trajectory is encoded by an adequate parametric representation. The optimizer generates an energy-optimal trajectory by efficiently using the robot kinematic redundancy while taking into account maximal joint motion, collision avoidance and local minima. To enable the resulting method to work in real-time, examples of the global planner are generalized using nearest neighbour approaches, Support Vector Machines and Gaussian process regression, which are compared in this context. Evaluations indicate that the presented method is highly efficient in complex tasks such as ball-catching.**

## I. Introduction

The robot catching task can be seen as a simple point-to-point control problem, solvable with inverse kinematics and interpolation in real-time [1]. If the solutions want to be in some way improved, then the task becomes a complex optimal control problem. This paper presents a careful analysis and empirical evaluation of the issues involved in nonlinear optimization for solving the catching task in an optimal way in real-time. A robot manipulator is considered to accomplish the task and it is assumed that its end-effector is always successful in grasping the target. Our focus is on the motion needed to reach the grasping point on the target trajectory.

There are several key issues of interest: firstly, optimal solutions for the given problem can rarely be obtained by running the optimization algorithm on line, as it is computationally too expensive and, due to local minima, it may not even converge to a good solution. Secondly, the optimization method should allow for realistic problems to be addressed, which requires treating important constraints on the movement such as collision avoidance and maximal velocities.

The optimal real-time planning problem is first formulated as a parametric nonlinear optimization problem. The joint positions are parameterized in time using a representation such as B-splines or trapezoidal functions. Inequality box constraints on joint position, velocity and actuation torque, as well as collision avoidance constraints, are included. Two different catching strategies are realized: the static catch, where the end-effector reaches the target trajectory and stops, and the dynamic catch, where the end-effector catches the target with some velocity in order to minimize the impact. As an example for a cost function, the energy is chosen, which brings the dynamics of the system clearly into play.

As the task of catching a moving target implies hard time constraints, a method is developed to obtain the optimal solutions in real-time. To ensure that the local optimizer starts with a good initial solution, globally optimal solutions are pre-computed offline for different initial target trajectories. In this paper, we evaluate several different approaches to using these initial solutions as starting points for the local search procedure. The first approach is to use look-up tables for determining good starting parameters, e.g., using different versions of $k$-nearest neighbours [18] with $k \in \{1, ..., 4\}$, while non-parametric methods such as Support Vector Regression (SVR) [19], [20] and Gaussian process regression (GPR) [14], are evaluated as alternatives with improved generalization. These approaches basically provide mappings between the three parameters which describe the target trajectory and the optimization parameters which describe the optimal solution.

The paper is then structured as follows: the rest of Section I presents a literature survey and the problem statement, while Section II describes the formulation of the optimization problem and Section III the method for solving the catching task in real-time. Section IV analyses the results and Section V includes a discussion and the conclusions. The adopted notation is such that all vector quantities are written in bold and are expressed in the inertial frame of reference.

### A. Related Work

The minimum energy problem for a non-redundant 6 DoF manipulator executing point-to-point maneuvers in configuration space was treated in [2], including collision avoidance. The resulting constrained boundary value problem was solved with direct single shooting. In [3] similar minimum energy problems were addressed, where direct collocation and indirect optimization were used instead. In [4], motion optimization was addressed for the kick motion of a humanoid robot, while also minimizing the energy. In [5], trajectory optimization was also used to solve robot motion tasks, however defined in Cartesian space rather than in configuration space.

Note however that in all of the works above, the real-time issue is not addressed. Furthermore, the optimization problem considered here is different from those formulated,

R. Lampariello, C. Castellini and G. Hirzinger are with the Institute of Robotics and Mechatronics (DLR), 82234 Weßling, Germany, email: *firstname.lastname@dlr.de*. D. Nguyen-Tuong and J. Peters are with the Max Planck Institute for Biological Cybernetics, 72076 Tübingen, Germany, email: *firstname.lastname@tuebingen.mpg.de*

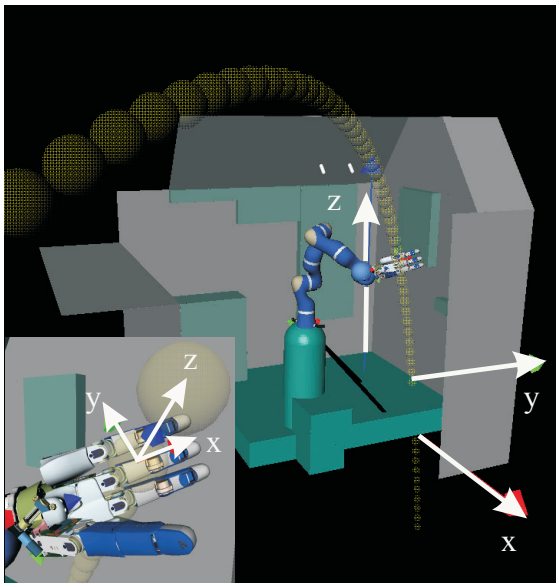


Fig. 1. The DLR ball-catching scenario with the ball-trajectory prediction trace, the LBR robot in catching configuration and a zoomed in picture of the hand, with the inertial and end-effector frames shown.

for e.g. in [2] and [3], in that: firstly, the final robot configuration is not given; secondly, the final time is not fixed; thirdly, there is a high kinematic redundancy resulting from using a 7 DoF robot and for the given task. The first point adds a set of nonlinear equality constraints, while the second and third add complexity, due to a resulting increase in local minima. These issues are also found in the problem dealt with in [4], where however local minima are not addressed. Collocation was not considered here as it requires a larger number of optimization parameters and is hence less suited for real-time application. In [7] a similar formulation of the nonlinear optimization problem is solved in a real-time setting, with a parallel multi start search and a low-dimensional search space.

Despite all the work on robot catching, e.g., as reviewed in [6], we have not found a methodological approach in the literature as we present here. As comparison, it is worth noting that humans perform catching movements as smooth point-to-point trajectories with bell-shaped velocity profiles and zero boundary velocities and accelerations [6].

In the learning literature, the generalization of trajectories has been suggested, for e.g., in [15], [16] and [17]. These approaches are complementary to the setup presented in this paper but differ significantly in scope and functionality. Their aim was the generalization of trajectories through regression which is only a necessary step for our aim to make nonlinear optimal control approaches feasible in real-time. They use artificial data [16] or kinesthetically recorded data [15] [17] which cover only a small space of the range of possible movements and cannot generalize beyond these. Here, we try to find a large set of globally optimal plans, generalize among them and ensure continued optimality by local optimization.

### B. Problem statement

The addressed problem is to develop a motion planner for catching a flying target, whose rotational motion is irrelevant (small, spherical, e.g., a ball), by means of a 7 DoF robot manipulator with rotational joints and rigid links. The test-bed is a simulation model of the DLR light-weight robot, shown in Fig. 1. It is assumed that the hand closing movement is always successful in grasping the target. Joint friction and elasticity are neglected. The target trajectory is assumed to be determined by a vision system (e.g., see [1]). The initial configuration of the robot is fixed. Trajectories should be found which bring the end-effector into an orientation suitable for grasping, i.e. such that the target velocity vector is at some predefined angle to it. The interception point is also determined by the motion planner. These requirements result in three equality constraints on the end-effector position and two equality constraints on the end-effector orientation. Due to the fact that the LBR robot has seven joints, a redundancy of degree two for the end configuration follows. The trajectory's duration and final configuration are open parameters determined by the optimizer.

The starting point of the target trajectory in Cartesian space is fixed. Its distance from the robot base is approx. 5 meters. Hence, with the speed resulting from this distance (and from limitations on throwing height due to the room ceiling and throwing velocity), the flight time will be approximately one second.

## II. Formulation of the Constrained Optimization Problem

The motion planning problem at hand contains a known obstacle region $O$ and a configuration space $C$ of dimensions $C(\boldsymbol{\theta}) \subseteq \Re^n$, with $n = 7$ and where $\boldsymbol{\theta}$ is the vector of robot joint positions. The time interval is unbounded: $t = [0, \infty)$. The robot system is fully actuated and therefore subject to a bounded action $\boldsymbol{\tau} \in \Re^n$, which is related to the system state $[\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}]$ by the state transition equation and where $\boldsymbol{\tau}$ is the vector of robot joint torques.

The nonlinear optimization problem can then be formulated as follows:

$$\min_{t_f, \boldsymbol{\theta}(t)} \Gamma(\boldsymbol{\theta}(t), \boldsymbol{\tau}(t), t_f) \tag{1}$$

subject to

$$\mathbf{M}(\boldsymbol{\theta})\,\ddot{\boldsymbol{\theta}}(t) + \mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\,\dot{\boldsymbol{\theta}}(t) + \mathbf{g}(\boldsymbol{\theta}) = \boldsymbol{\tau}, \tag{2}$$

$$\mathbf{h}(t_f, \boldsymbol{\theta}(t)) \leq 0, \tag{3}$$

$$\mathbf{h}_{\text{coll}}(t_f, \boldsymbol{\theta}(t)) \leq 0, \tag{4}$$

$$\mathbf{g}(\mathbf{r}^e(t_f), \boldsymbol{\phi}^e(t_f)) = 0, \tag{5}$$

$$\boldsymbol{\theta}(0) = \boldsymbol{\theta}_{\text{in}}, \dot{\boldsymbol{\theta}}(0) = 0, \dot{\boldsymbol{\theta}}(t_f) = 0. \tag{6}$$

for $0 \leq t \leq t_f$ and where $t_f$ is the final time, $\Gamma$ is a predefined cost function, $\mathbf{h}$ are inequality box constraints of type $\mathbf{x}_{\text{min}} \leq \mathbf{x}(t) \leq \mathbf{x}_{\text{max}}$, for $\mathbf{x} = \{\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \boldsymbol{\tau}\}$ and $\mathbf{h}_{\text{coll}}$ are collision avoidance constraints. Eqn. (2) express

the state transition equation of the robot. The functions $\mathbf{g}(\mathbf{r}^e(t_f), \boldsymbol{\phi}^e(t_f))$ are five equality constraints on the final end-effector state $[\mathbf{r}^e, \boldsymbol{\phi}^e]$ (see Section II-C). Finally, Eq. (6) expresses boundary conditions on position, where $\boldsymbol{\theta}_{\text{in}}$ is the given initial configuration, and on velocity. More details on the boundary conditions on acceleration and jerk will be given in Section III-A.

In the following sections we will address the formulations of the cost function, of the inequality constraints and of the equality constraints.

### A. Cost function

The chosen cost function is the mechanical energy. This is a classical choice to improve the energy consumption of the given system (note that the LBR has a very high efficiency in energy dissipation, therefore we neglected the latter). It also adds a strong dynamics-dependent element to the optimization problem, as opposed, for example, to minimum distance.

The mechanical energy is computed here as follows (similarly to [2] and [3]):

$$\Gamma_{\text{energy}}(\mathbf{p}) = \int_0^{t_f} (\boldsymbol{\tau}^T(t)\,\dot{\boldsymbol{\theta}}(t))^2 \;+\; \|\mathbf{J}_{\text{motor}}\,\dot{\boldsymbol{\theta}}(t)\|^2 dt\,, \quad (7)$$

where the first term is the integral of the power for a robot model which neglects the joint motor inertias and the second term represents the motor kinetic energy, which is of comparable size to the first (not included in [2] and [3]). The symbol $\mathbf{J}_{\text{motor}} = \text{diag}(J_1 N_1, ..., J_n N_n)$ expresses an $(n \times n)$ diagonal matrix with elements whose $i^{\text{th}}$ diagonal element is the $i^{\text{th}}$ motor inertia $J_i$ multiplied by the $i^{\text{th}}$ gear reduction ratio $N_i$.

### B. Inequality constraints

The bounds of the box constraints are given by the robot design specifications, e.g., due to joint limits and maximal joint velocities. Further constraints arise from the collision avoidance both with the environment and of the robot with itself. To detect collision and to formulate the collision avoidance problem within a nonlinear programming context, bodies are represented here as convex polytopes. For this purpose, these bodies consist of capsules to represent the robot links and the end-effector, and of a box to represent an obstacle in the robot workspace (Note that a capsule is similar to a normal cylinder except that it has half-sphere caps at its ends.). For these types of bodies, it is possible to efficiently compute, in case of collision, the penetration depth as the minimal length of translation needed to separate them.

The collision avoidance problem can be formulated straightforwardly as a set of inequality constraints in the optimization problem:

$$D(i) \;>\; 0.0, \quad 1 \;\leq\; i \;\leq\; m_{\text{coll}}, \quad (8)$$

where the function $D(i)$ constitutes a minimum distance between two bodies or a penetration depth, if the two bodies intersect. The scalar $m_{\text{coll}}$ is the number of body pairs in the given problem.

### C. Equality constraints

Additional equality constraints are required on the final end-effector position and orientation, in order for it to meet the target at some point on the trajectory. A distinction is introduced between the static and the dynamic catches.

*1) Static catch:* In this case, the end-effector arrives at the grasping point with zero end velocity and the equality constraint is formulated as follows:

$$\mathbf{r}^e(t_f, \mathbf{p}) - \mathbf{r}^{\text{target}}(t_f) = 0, \quad (9)$$

$$\boldsymbol{\phi}^e(t_f, \mathbf{p}) - \boldsymbol{\phi}^{\text{target}}(t_f)) = 0, \quad (10)$$

where $\mathbf{r}^e$ is the end-effector position vector, computed at the final time $t_f = p(N)$, $\mathbf{r}^{\text{target}}$ is the given target position vector at the final time, $\boldsymbol{\phi}^e$ are the two angles which describe the direction of the -z axis of the end-effector (see Fig. 1) and $\boldsymbol{\phi}^{\text{target}}$ are the two angles which describe the direction of the target velocity vector, also computed at the final time. These constraints are nonlinear in the parameters $\mathbf{p}$ and reduce the open DoFs from 7 to 2.

*2) Dynamic catch:* In this case, in order to reduce the impact with the target, the end-effector moves in the same direction as the target. This effect can be achieved by imposing extra equality constraints on the end-effector position and orientation, of the type expressed in Eqs. (9),(10). The first constraint was imposed at a time $t_{\text{mid}} = t_f - \Delta t$, for $\Delta t < t_f$ as

$$\mathbf{r}^e(t_{\text{mid}}, \mathbf{p}) - \mathbf{r}^{\text{target}}(t_{\text{interceptTarget}}) = 0, \quad (11)$$

$$\boldsymbol{\theta}^e(t_{\text{mid}}, \mathbf{p}) - \boldsymbol{\phi}^{\text{target}}(t_{\text{interceptTarget}}) = 0. \quad (12)$$

The value of $\mathbf{r}^{\text{target}}$ and $\boldsymbol{\phi}^{\text{target}}$ in Eqs. (11), (12) is taken at a time $t_{\text{interceptTarget}} = t_f - k_{\text{mid}}\,\Delta t$, for $k_{\text{mid}} < 1$. Note that the robot is often not able to travel as fast as the target due to joint velocity limits. The second set of equality constraints was imposed at a time $t_{\text{mid}\,2} = t_f - \Delta t/2$, half way between the first extra constraint and the final constraint points.

## III. Efficient Motion Planning in Real-Time

In this section the methods to solve the optimization and the learning problems described above are addressed.

### A. Parameterization of the trajectories

Two parameterizations can be chosen for the joint states: a classical trapezoidal function and an order-4 B-spline. Both are described below. The order-4 B-spline was chosen in order to allow for smoothness up to the third derivative. The trapezoidal function was used to provide further means of comparison between low and high dimensional parameterization spaces.

*1) Order 4 B-spline:* We choose periodic uniform B-splines for their particularly compact matrix form. For $N$ vertices, $n_{\text{seg}} = N - 3$ segments of length $t_{\text{seg}} = t_f/(N - 3)$ result. It follows that for the internal time of the $i^{\text{th}}$ segment $u(t) = t/t_{\text{seg}} - (i-1)\,t_{\text{seg}}$, such that $0 \leq u < 1$, the computation of the uniform B-spline and

derivatives is given by (as in [10])

$$\begin{pmatrix} s_i(u) \\ \dot{s}_i(u) \\ \ddot{s}_i(u) \\ \dddot{s}_i(u) \end{pmatrix} = \frac{1}{6} C(u)\, A \begin{pmatrix} B_i \\ B_{i+1} \\ B_{i+2} \\ B_{i+3} \end{pmatrix} \text{ for } 1 \leq i \leq n_{\text{seg}}, \quad (13)$$

where $B_i$ represents the $i^{\text{th}}$ vertex, $A$ is a constant matrix and $C(u)$ the matrix of basis functions. Furthermore, these matrices are invertible, so that they can be used to satisfy the boundary conditions. These are given by

$$\begin{pmatrix} s_0(0) \\ \dot{s}_0(0) \\ \ddot{s}_0(0) \\ \dddot{s}_0(0) \end{pmatrix} = \begin{pmatrix} \boldsymbol{\theta}_{in} \\ 0 \\ 0 \\ \mathbf{p}_{\mathbf{j}_0} \end{pmatrix}, \begin{pmatrix} s_{N-3}(1) \\ \dot{s}_{N-3}(1) \\ \ddot{s}_{N-3}(1) \\ \dddot{s}_{N-3}(1) \end{pmatrix} = \begin{pmatrix} \mathbf{p}_{\boldsymbol{\theta}_{t_f}} \\ 0 \\ 0 \\ \mathbf{p}_{\mathbf{j}_{t_f}} \end{pmatrix}, \quad (14)$$

where $\mathbf{p}_{\mathbf{j}_0}$ are the parameters for the jerk at time $t = 0$, $\mathbf{p}_{\boldsymbol{\theta}_{t_f}}$ are the parameters for the joint positions and $\mathbf{p}_{\mathbf{j}_{t_f}}$ are the parameters for the jerk at time $t = t_f$. Note that of the 8 boundary conditions, 5 are predefined, including zero velocities and accelerations. It was chosen not to set the initial and final jerk to zero, as it would significantly reduce the family of curves available to the optimization.

*2) Classical trapezoidal velocity profile:* The trapezoidal velocity profile [11], entails three phases: a constant acceleration phase, a cruise velocity and constant deceleration phase. The first and last phases have the same time duration and the same gradient modulus, see Fig. (2).

Two parameters determine the profile: $t_1$ and $x_{t_f} = x(t_f)$. It follows that $t_2 = t_f - t_1$ and $\ddot{x}_0 = -\ddot{x}_2 = (x_0 - x_{tf})/(t_1^2 - t_f\, t_1)$. Note that $\ddot{x}_1 = 0$. The computation of the profile is thus straightforward. The optimization problem contains $7 \times 2$ parameters for the $n = 7$ joint states and 1 parameter for the end time $t_f$, in all 15 parameters.

### B. Search for the global optimum

The optimization problem presented in Section II is strongly limited by local mimima (see Section IV-B for examples). To overcome this problem, we run the optimization for a given target trajectory for 100 times, using different initial guesses for the starting parameters, chosen with the following procedure: a robot configuration $\boldsymbol{\theta}$ is defined randomly, within the range of allowed values; a trajectory is determined as a straight line between the given initial and the randomly defined configuration, by algebraic computations of the B-spline parameters; these latter parameters are taken as initial guess. Subsequently, the starting parameters which yield the best optimization result of the 100 trials is taken as global optimium.

### C. Offline method for the local constrained optimization problem

The optimization problem described above is solved as a nonlinear programming problem (NPL), by satisfying the equality and inequality constraints at a finite number of $k$ via points. The proposed optimization method is based on direct single shooting, with parameterization of the system independent states in time, as for e.g. in [2], i.e., $\boldsymbol{\theta} = \boldsymbol{\theta}(t, \mathbf{p})$ with $\mathbf{p} \subseteq \Re^N$, for $N$ optimization parameters, as described in Section III-A. The control forces are then computed from the state transition Eq. (2). The NPL is solved with the Sequential Quadratic Programming algorithm from the MOPS library [9].

To compute the penetration depth between two bodies, the ODE library was used [8]. The library allows representing objects as boxes or capsules. Each pair of intersecting objects is treated separately and penetration depth can be evaluated for each pair straightforwardly.

### D. Efficient Initialization of the Local Planner

The optimization method presented in Section III-C cannot be used on-line, since it takes a prohibitively long time to converge (only 60 milliseconds are available on-line for the computation) and is likely to get stuck into local minima. It is therefore paramount to choose a good initial guess of the $N$ parameter values depending on the estimated velocity of the moving object, and to do it quickly. In order to do that, we generate offline a set of (initial velocity, parameter values) pairs that we use as a *training set*; several machine learning regression methods are then compared in order to determine a map from the estimated velocity to the optimization parameters: k-Nearest Neighbours ($k$-NN) with $k = 1, 2, 3, 4$[1], Support Vector Machines (SVM) and Gaussian Process Regression (GPR). The input space has dimension 3 (the estimated values of the object velocity) and the output space is $N$, the number of parameters.

*1) $k$-Nearest Neighbour Regression:* A $k$-NN [18] is a simple local linear approximator of a function given a set of known (sample,target) pairs (the training set):

$$k\text{-NN}(\mathbf{x}) = \sum_{i \in I_{\mathbf{x}}} \alpha_i t_i$$

where $I_{\mathbf{x}}$ is the set of the indices of the $k$ $\mathbf{x}_i$s which have minimum Euclidean distance from $\mathbf{x}$ in the chosen training set, $t_i$ is the target value associated to $\mathbf{x}_i$ and, in our case,

$$\alpha_i = \frac{||\mathbf{x} - \mathbf{x}_i||^{-2}}{\sum_{i \in I_{\mathbf{x}}} ||\mathbf{x} - \mathbf{x}_i||^{-2}}.$$

(This particular choice of the $\alpha_i$s is called Inverse-distance-weighted $k$-NN.) Notice that 1-NN is equivalent to a look-up table, that is, probably the simplest way of solving this problem.

*2) Support Vector Regression:* Support Vector Regression [19], [20] builds a map between an input space and an output space as a weighted sum of basic functions induced by the a-priori choice of a *kernel*. In our case we have chosen, as is rather customary, a Gaussian kernel, so that the solution to the problem is

$$\text{SVM}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i G(\mathbf{x}_i, \sigma)$$

[1]We initially determined that no relevant advantage was obtained with $k > 4$.

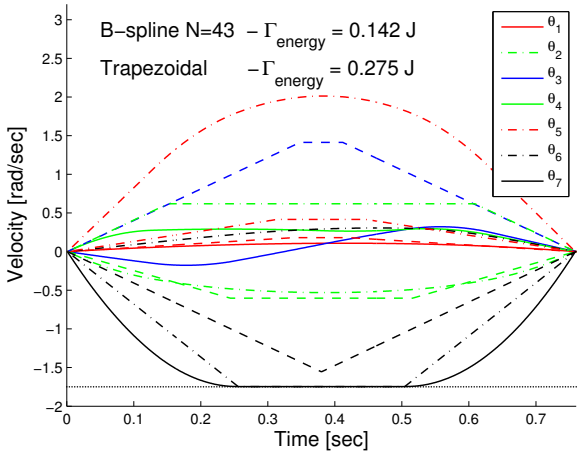


Fig. 2. Robot joint velocity profiles for a static catch: B-spline (solid/dashdot) and trapezoidal (dashed/dashdot) profiles shown. Velocity limit shown for the 6th and 7th joints. A comparison between the cost obtained with the two trajectory parameterizations is also given.

where $N$ is the number of samples in the training set, $G(\boldsymbol{\mu}, \sigma)$ is a Gaussian function with mean value $\boldsymbol{\mu}$ and covariance $\sigma 2$, and the $\alpha_i$s are determined by solving a regularised quadratic optimization problem in which a quantity called $C$ must be fixed a priori. $C, \sigma$ have been found in an initial round of experiments via cross-validation and grid-search.

*3) Gaussian Process Regression:* Gaussian Process Regression [14] is also a probabilistic method to approximate a functional mapping. To predict a point $\mathbf{x}_*$ we evaluate the conditional mean of a Gaussian process model, given by

$$\bar{f}(\mathbf{x}_*) = \mathbf{k}_*{}^T \left(\mathbf{K} + \sigma_n 2\mathbf{I}\right)^{-1} \mathbf{y} = \mathbf{k}_*{}^T \boldsymbol{\alpha}\,, \qquad (15)$$

where $\mathbf{k}_*$ is a kernel vector evaluated on the query point and training inputs, $\mathbf{K}$ is the kernel matrix, $\mathbf{y}$ is the target vector and $\sigma_n 2$ is the noise variance. The open parameters of a GP model are optimized using the available data. To make GPR feasible for a real-time application, we compute the prediction vector $\boldsymbol{\alpha}$ off-line (this includes an expensive matrix inversion) and then during prediction only the covariance vector $\mathbf{k}_*$ with the pre-computed $\boldsymbol{\alpha}$ is evaluated.

### E. Real-time implementation

In the real-time setting, an initial guess for the optimized robot trajectory parameters is first computed for the target trajectory at hand, by means of the chosen learning method (between those described in Section III-D). Subsequently, based on this initial guess, the motion planner is run on-line to compute a successful robot trajectory. For this on-line version of the motion planner, no cost function is optimized, and the number of via points is greatly reduced, such that the computational time for its execution is sufficiently short. The on-line planner however still satisfies the equality and inequality constraints defined in Sections II-B and II-C, thus adjusting any small discrepancy between the trajectory which results from the initial guess and the desired trajectory.

The issue of collision avoidance in real-time was however not addressed here. It may be assumed that training points representing collision-free solutions would be generated with conservative sizes of the representative polytopes in the problem at hand. This way, the likelihood of a collision occurring due to on-line trajectory corrections is minimal and the resulting inequality constraints may well be handled in a sufficient computational run time (a collision detection function call was measured to last $8e10^{-7}$ seconds). In order to investigate this issue a relevant set of training data must be generated for a case with potential collisions, e.g. with an obstacle in the workspace.

## IV. Analysis of Results

The proposed method is applied in simulation for the ball catching scenario, shown in Fig. 1. Following are detailed examples to demonstrate its effectiveness.

### A. Catching a flying target: static and dynamic catch

As also hinted by the theory (see [11]), the best energy optimal solutions were found to be those with non-zero accelerations at the boundaries. We however chose to set them to zero, to avoid large jerk values.

When comparing the cost $\Gamma_{\text{energy}}$, defined in Eq. (7), for different number of parameters $N$, very little improvement could be found. Parameterizations with $N$=43, 71 and 141 were compared, resulting in 6, 10 and 20 parameters per state respectively. A sensible number for $N$ was then taken to be 43. A comparison with the trapezoidal parameterization reveals that, for the cost function $\Gamma_{\text{energy}}$, the loss can be very pronounced: for the example in Fig. 2 the difference was found to be 48%.

For the off-line computations, the number of via points was set to $k = 500$. All runs were first performed with an accuracy of $10^{-8}$ and in a second iteration with accuracy $10^{-12}$ (a first iteration is completed when the optimization is run once with a given initial guess; a second iteration is a new run of the optimization with the initial guess given by the result of the first iteration). No more than two iterations were performed.

In the solutions for the static grasp, one can distinctively see that the velocity constraints play an important role. For example, in Fig. 2, the bottom curve evidently meets a constraint at -1.75 rad/sec. Solutions often resemble the parabolic profile described by the theory [11]. However, due to the inequality constraints and the complex nonlinear robot kinematics, the parabolic profiles are often distorted and sometimes not even recognizable.

The implementation of the collision avoidance was applied to the self-collision of the robot and to the collision with an obstacle in the robot workspace. The resulting number of body pairs was optimized to $m_{\text{coll}} = 16$ for the eight bodies. However self-collision was found to never occur, after the minimization of the cost function and for the given initial configuration. When introducing the

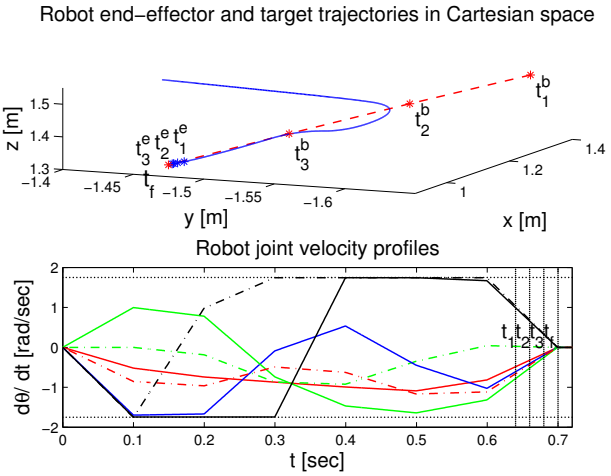


Fig. 3. Example of a dynamic catch. Top: end-effector (solid blue) and target (dotted red) trajectories; four equal time intervals shown ($t_1^b = t_1^e, ...$). Bottom: Robot joint velocity profiles with saturation limits at $\pm$ 1.75 rad/sec. and same four time intervals shown ($t_1 = t_1^e, ...$).

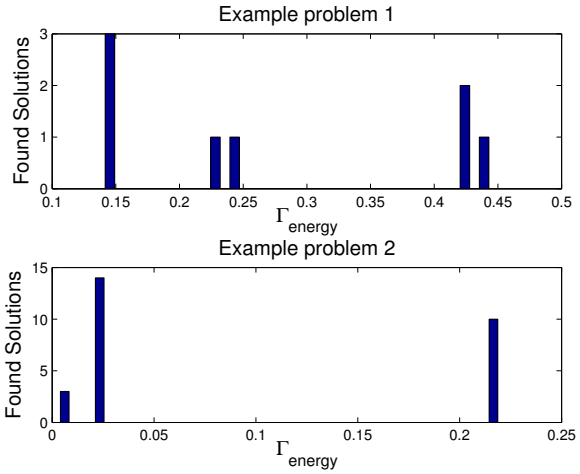


Fig. 4. Histograms showing the found solutions for two examples of a static catch, attempted each 100 times with a random bounded initial guess: the different found solutions show the presence of local minima.

obstacle of dimensions [0.2, 0.2, 4] meters at a position $\mathbf{x}_{obst} = (1.6, -1.1, 0)$, giving rise to collisions, the planner successfully found collision-free trajectories, while minimizing the energy cost function. In doing so, the capsule representing the end-effector and the box representing the obstacle may be in contact, but without overlapping, for a substantial portion of the motion.

Fig. 2 shows an example of a static grasp, for which the velocity at the time $t = t_f$, when the end-effector meets the target, is zero. Fig. 3 instead shows an example in which the robot meets the target with a non-zero velocity. The value of the parameters defined in Section II-C was chosen by manual tuning to be $\Delta t = 0.45$ and $k_{\text{mid}} = 0.08$ respectively (these may be added as optimization parameters at a later stage). The velocity of the end-effector between times $t_1^e$ and $t_f$ is 5% of the target velocity. The reduction of the impact with the target was however strongly limited by the joint velocity constraints, which can be seen in the bottom graph to be met for most of the motion. In the top figure, $t_3^b$ is approximately coincident with $t_{\text{mid}}$.

### B. Global Optimality of Solutions

Fig. 4 shows two typical histograms, for two example static catching problems. We repeated the runs 100 times for each problem, as described in Section III-B. The histograms include all the successful runs, for which all equality and inequality constraints were satisfied. It is evident that local minima exist, as shown by the different found solutions in each problem. Also note that in both cases, only a small percentage of the 100 runs converged to a solution, which gives clear evidence of the strong dependence of the convergence on a good initial guess (for the dynamic catch this dependence is expected to be even stronger).

After inspection, it was found that the highly nonlinear robot kinematics is likely to cause the local minima, which can be distinguished clearly due to their different final configurations. As an example, two end configurations for the top case of Fig. 4 are shown in table I.

TABLE I

TWO END CONFIGURATIONS FOR A STATIC CATCH EXAMPLE SHOWING TWO LOCAL MINIMA

| $\boldsymbol{\theta}(t_f)$ [deg] | | | | | | | $\boldsymbol{\Gamma}$[J] |
|---|---|---|---|---|---|---|---|
| -129 | -24 | 152 | -21 | 33 | 47 | 131 | 0.14 |
| -56 | 93 | 68 | 59 | -81 | 38 | 72 | 0.44 |

### C. Comparing machine learning methods

A training data set was first generated for the static catch problem with the B-splines parameterization. The three target velocity vector components were first sampled at regular intervals from a range of values for which catching solutions may exist. For each point, the global optimum was sought, as described in Section III-B, to produce a first coarse grid. The boundaries of this grid were then expanded until no more solutions were found. More midpoints of the grid were then also computed with the initial guess taken from the coarse grid, to produce a finer grid, of sufficient fineness for the subsequent learning process. The resulting data set contains 1825 (sample, target) pairs, with a spacing in the target velocity space of [0.17,0.25,0.125] m/s in the range [-2,-6],[2,5],[2.5,5.5] m/s.

The total computation time of the data set on an Intel Xeon CPU W3520 2.67GHz machine is in the order of magnitude of 100 hours, which makes it unfeasible for online optimization. It is also desirable to find a method which works with fewer samples, at the same time keeping a reasonable error rate. In order to analyze this trade-off each method described in Section III-D is trained on 5 sets consisting of $900, 700, 500, 300, 100$ pairs drawn from the original dataset, chosen in order to be geometrically uniformly spaced. Each method is then tested on the samples not used for training.

TABLE II

AVERAGE COST INCREASE (%) W.R.T. GLOBAL MINIMUM COST FOR EACH METHOD AND TRAINING SET SIZE.

| | **98** | **302** | **504** | **700** | **900** | **1825** |
|---|---|---|---|---|---|---|
| **1-NN** | 187 | 113 | 112 | 103 | 93 | 46 |
| **2-NN** | 118 | 57 | 56 | 71 | 63 | 29 |
| **3-NN** | 105 | 55 | 52 | 58 | 53 | 23 |
| **4-NN** | 99 | 50 | 41 | 57 | 46 | 26 |
| **SVM** | 92 | 51 | 41 | 36 | 30 | 21 |
| **GPR** | 94 | 36 | 33 | 21 | 19 | 11 |

TABLE III

RUNS BELOW 60MS (%) FOR EACH METHOD AND TRAINING SET SIZE.

| | **98** | **302** | **504** | **700** | **900** | **1825** |
|---|---|---|---|---|---|---|
| **1-NN** | 91 | 95 | 94 | 94 | 95 | 96 |
| **2-NN** | 93 | 94 | 95 | 94 | 96 | 94 |
| **3-NN** | 94 | 95 | 95 | 95 | 96 | 95 |
| **4-NN** | 93 | 96 | 95 | 95 | 96 | 96 |
| **SVM** | 93 | 94 | 94 | 95 | 95 | 95 |
| **GPR** | 93 | 94 | 93 | 90 | 89 | 31 |

Figure 5 shows the results. The chosen error measure is the ratio of the Mean-Squared-Error obtained for each testing set and the variance of the target trajectory parameter values for the same test set.

First of all, by considering all panels together, one can see that the error rates are similar inter-joint. For example, the joint parameters $31-36$, corresponding to joint #6, are consistently harder to guess than, say, parameters $25-30$, corresponding to joint #5. This diversity depends on the setup and the geometry of the catching movement. Secondly, notice that 1-NN consistently shows a worse error rate than all other methods. Evidently, something slightly more complex than a simple look-up table is required, if smaller training sets want to be used. Overall, GPR is the best method in most cases, and it is so consistently across datasets for joints #2 (parameters 7-13), #4 (parameters 19-25) and #7 (parameters 37-42).

### *D. Real-time implementation*

Lastly, we implemented the methods described above in the real-time simulation environment, for a more thorough evaluation. (We also show the performance obtained by each method trained on the full data set of $1825$ samples; this could not be done in the previous Section since no testing set is available in this case.) For each method, we firstly considered the optimizer convergence success rate (solved NPL problems / total trials), randomly choosing $900$ target velocities within the feasible range. Essentially all methods performed equally well, with success rates between 95% and 98%. Surprisingly, decreasing the training set size does not affect this performance index, indicating that a relatively large error in prediction is tolerated.

A rather different scenario appears when we turn to stricter performance measures. Tables II and III show, in turn, the average percentual cost increase with respect to the average global minimum cost, and the percentage of runs below 60ms (again, $900$ random target velocity values were generated). Note that the number of via points was set to $k = 10$ and the optimization accuracy to $10^{-3}$. Also note that these run times are sufficient to accommodate for multiple corrections of the target trajectory, which may arise from new updates of the vision system (as done in [7]). This was verified in the LBR ball-catching simulation environment.

Consider Table II: clearly, as the training set is reduced, the energy consumption increases; also, $k$-NN perform better as $k$ is increased; SVM is better and GPR is the best. If the main requirement is to spare energy then, GPR should be used since it will increase the energy consumption only by $11\%$ with $1825$ samples. When the training set is reduced to $98$ samples though, their performance becomes similar to that of SVM.

Consider now Table III: all methods keep the required time at an acceptably low value. Here however, GPR suffers from a decrease in performance as the training set becomes larger, keeping the pace in only $31\%$ of the cases when the set is full ($1825$ samples).

## V. DISCUSSION AND CONCLUSION

In this section the results of this work are briefly discuss and final conclusions are given.

### *A. Discussion*

From the results presented in tables II and III it is evident that for the on-line implementation a trade-off needs to be made between average cost increase, computational time and number of training points. Particularly for the ball catching task and with the LBR robot (and its joint velocity limits), the computational time sets a hard constraint which must be fulfilled at the expense of average loss in cost function. Noticeable improvement with respect to a 1 -NN with 1825 training points can be seen, for e.g., with a 4-NN and 500 points or a GPR and 300 points. The GPR also gives the best performance in terms of parameter prediction and cost increase, but at a higher computational cost. This problem is however only critical for tasks for which the computational time is very limited. The method may turn ideal for tasks which allow more computational time during real-time performance, for which some degree of on-line optimization may even be possible.

The authors are aware of the fact that the presence of nondiffentiable points in the penetration depth function, may give rise to numerical problems for the gradient-based optimization, which requires $\mathbf{C}^2$ smooth objective and constraint functions in order to converge. However, the results found by the authors with a preliminary analysis in implementing the ODE collision detection function (as described in section II-B) to a multitude of test cases with static obstacles in 2D, were positive. More mathematically sound methods can be found in the literature (see [12], [13]). Particularly in [13], this problem is partially solved by introducing a method to generate strictly convex hulls,

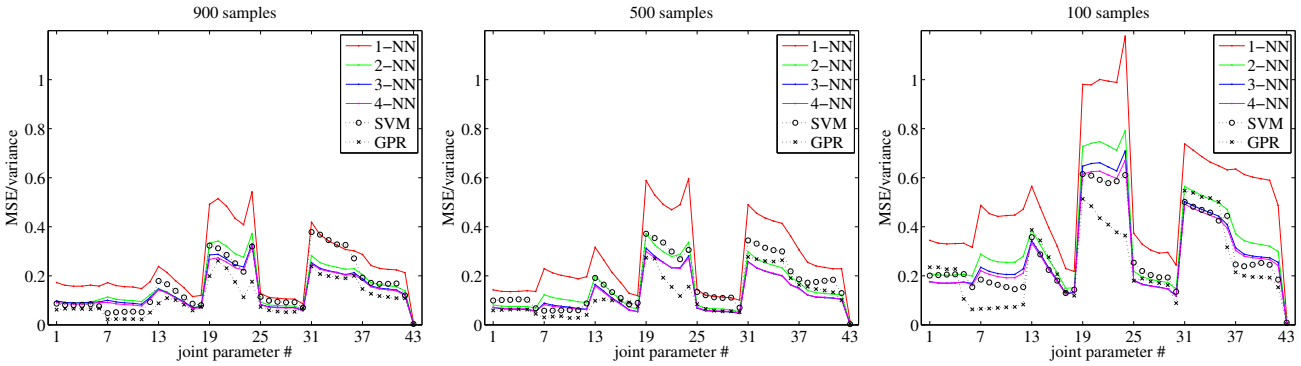


Fig. 5. Comparison of Machine Learning methods for training sets of decreasing size.

for which the discontinuities only remain for the case of deep penetrations.

As described in Section III-E, further work will aim at establishing the speed of convergence of the on-line motion planner in the presence of collisions. If this will introduce a critical time factor, the size of the representative polytopes will need to be made more conservative, clearly at the loss of the cost function optimization (e.g., in order to avoid collision, the ball will be caught at a less optimal point of its trajectory).

### B. Conclusion

The problem of catching a small flying object was addressed using a nonlinear optimization framework. A suitable parameterization was implemented with B-splines and a comparison made to a more simple trapezoidal function. Evidence was given that much more efficient solutions can be found with the former. The capability of handling collision avoidance constraints was also demonstrated for the off-line generation of optimal solutions. Subsequently, a methodology was described for searching global solutions, thus overcoming the problem of local minima.

Finally different methods to provide the computationally expensive optimal solutions on-line were applied to the ball-catching problem in simulation. Due to the hard constraint on the final time, it was found that some performance in the accuracy of the prediction has to be sacrificed. A substantial improvement with respect to a look-up table approach was shown.

The presented method clearly has the limitation of working for a fixed initial robot configuration and a fixed starting point of the target trajectory. Furthermore, grasping of larger targets, for which the rotational motion is also important, will necessarily require six parameters rather than three to represent their trajectories. Future work will concentrate on these difficult issues. However, for the relatively simple addressed problem, it was shown that the solutions computed with nonlinear optimization can be used on-line, to a noticeable advantage of the here arbitrarily chosen cost function.

## REFERENCES


[1] U. Frese, B. Bäuml, S. Haidacher, G. Schreiber, I. Schaefer, M. Hähnle and G. Hirzinger, "Off-the-Shelf Vision for a Robotic Ball Catcher", *Proceedings of IEEE International Conference on Intelligent RObots and Systems (IROS)*, 2001.

[2] J. Park: "Convergence properties of gradient-based numerical motion-optimizations for manipulator arms amid static or moving obstacles", *Robotica*, Vol.22, pp. 649-659, 2004.

[3] O. v. Stryk, M. Schlemmer: "Optimal control of the industrial robot Manutec r3". In: R. Bulirsch, D. Kraft (eds.): Computational Optimal Control. International Series of Numerical Mathematics 115 (Basel: Birkhuser, 1994), pp. 367-382, 1994.

[4] S. Miossec, K. Yokoi and A. Kheddar, "Development of a software for motion optimization of robots - Application to the kick motion of the HRP-2 robot", *Proceedings of IEEE International Conference on Robotics and Biomimetics*, 2006.

[5] K. Harada, K. Hauser, T. Bretl and J.-C. Latombe, "Natural motion generation for humanoid robots", *Proceedings of IEEE International Conference on Intelligent RObots and Systems (IROS)*, 2006.

[6] M. Riley and C. Atkeson, "Robot catching: Towards engaging human-humanoid interaction", *Auton. Robots*, Vol. 12, pp.119-128, 2002.

[7] B. Bäuml, T. Wimböck and G. Hirzinger, "Kinematically Optimal Catching a Flying Ball with a Hand-Arm-System", *Proceedings of IEEE International Conference on Intelligent RObots and Systems (IROS)*, 2010.

[8] http://opende.sourceforge.net/wiki/index.php/Manual_(Collision_Detection)

[9] H.-D. Joos, "MOPS Multi-Objective Parameter Synthesis User's Guide V 5.3", DLR-Interner Bericht, DLR IB 515-08-37, 2008.

[10] D.F. Rogers and J.A. Adams, *Mathematical Elements for Computer Graphics*, McGraw-Hill Publishing Company, Second Edition, 1990.

[11] L. Sciavicco and B. Siciliano, *Modeling and Control of Robot Manipulators*, McGraw-Hill Pubblishing Company, 1996.

[12] C.J. Ong and E. Gilbert, "Robot Path Planning with Penetration Growth Distance", *Journal of Robotic Systems*, vol. 15, no. 2, pp. 57-74, 1998.

[13] A. Escande, S. Miossec and A. Kheddar, "Continuous gradient proximity distance for humanoids free-collision optimized-postures", *Proceedings of International Conference on Humanoid Robots*, 2007.

[14] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT-Press, 2006.

[15] Nikolay Jetchev, Marc Toussaint (2009): Trajectory Prediction: Learning to Map Situations to Robot Trajectories. 25th International Conference on Machine Learning (ICML 2009).

[16] J. Peters, J. Kober, K. Muelling, D. Nguyen-Tuong and O. Kroemer, (2009). Towards Motor Skill Learning for Robotics, Proceedings of the International Symposium on Robotics Research (ISRR).

[17] Ude, Ales; Gams, Andrej; Asfour, Tamim; Morimoto, Jun (2010). Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives. IEEE Transactions on Robotics.

[18] T. Cover and P. Hart, "Nearest neighbor pattern classification", *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, 1967.

[19] B.E. Boser, I.M. Guyon and V.N. Vapnik, "A training algorithm for optimal margin classifiers", *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, ACM press, pp. 144-152, 1992.

[20] Alex J. Smola and B. Schölkopf, "A tutorial on Support Vector Regression", *Statistics and Computing*, vol. 14, no. 3, 199-222, 2004.