Efficient Trajectory Reshaping in a Dynamic Environment

Martin Biel
Dept. of Automatic Control, Electrical Engineering
KTH, Royal Institute of Technology
Stockholm, Sweden
mbiel@kth.se

Mikael Norrlöf
Dept. of E.E., Linköping University
Linköping, Sweden
Robotics and Motion Division, ABB AB, Västerås, Sweden, mikael.norrlof@liu.se

***Abstract*—A general trajectory planner for optimal control problems is presented and applied to a robot system. The approach is based on timed elastic bands and nonlinear model predictive control. By exploiting the sparsity in the underlying optimization problems the computational effort can be significantly reduced, resulting in a real-time capable planner. In addition, a localization based switching strategy is employed to enforce convergence and stability. The planning procedure is illustrated in a robotics application using a realistic SCARA type robot.**

I. Introduction

Real-time trajectory planning is a fundamental part of motion control for robotic systems. Efficient optimization formulations are typically used to achieve a high degree of autonomy of the system. Generally, a multi-objective problem is posed to form a trade-off between different aspects, such as time, energy, safety or system lifetime. Optimal trajectories can be generated by first performing path planning to find a geometric path, and then in a second step determine an optimal trajectory along that path. The approach is referred to as the decoupled approach [1]–[3] and in [4] it is shown that the trajectory generation can be formulated as a convex optimization problem which can be solved efficiently.

In many industrial applications the path cannot be decoupled from the trajectory generation. For instance, the targets can be moving on a conveyor which is not controlled by the robot controller, obstacles may also be present in the workspace of the robot. Therefore, a combined approach for trajectory optimization is required. In [5], a technique is presented where a given trajectory is dynamically updated based on sensor input through a so called elastic band framework. A trajectory that connects the start state to the end state generates an initial elastic band, which then deforms around obstacles by inferring virtual forces on the band. The technique enables real-time collision avoidance while also keeping the geometric path as short as possible. However, the system dynamics cannot easily be included in the trajectory planning.

The concept of elastic bands was extended in [6] and [7], where the time interval between poses is included to form a so called *Timed Elastic Band* (TEB). The inclusion of temporal information makes it possible to account for system dynamics and optimize with respect to time. The trajectory planning problem is modeled as a multi-objective optimization problem, where collision avoidance and system dynamics are introduced through penalty functions in the objective while also aiming to minimize the transition time. Most recently, in [8], the timed elastic band concept was applied in an MPC setting. Compared to [8], the present work addresses the real time performance aspect and includes variations in the end target.

The main contribution in this work is a general framework for online trajectory optimization of nonlinear systems that operate in dynamic environments. The reshaping strategy is an extension of [8] and timed elastic band framework is combined with non-linear model predictive control (NMPC) techniques to achieve quasi time-optimal trajectories. In addition the sparsity in the underlying optimization problems is exploited. This allows for computationally efficient implementation of the solution. A novel switching strategy is also employed to enforce convergence and stability as the system is driven to some target state.

Section II outlines the trajectory planning procedure. Section III presents a brief summary of the software implementation of the framework, followed by Section IV where the planner is employed in various simulation examples. Finally, Section V makes conclusions and lists possible future work.

II. Trajectory Planning Procedure

A. Problem Formulation

In the general setting, the aim is to determine a time-optimal trajectory $\boldsymbol{y}(t)$, given the system dynamics, and compute the control signal $\boldsymbol{u}(t)$ necessary to realize the trajectory. The task is here represented as an optimal control problem,

$$\min_{\boldsymbol{u}(.)} t_f \text{ s.t. } \begin{cases} \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \\ \boldsymbol{y}(t) = g(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \boldsymbol{x}(t) \in \mathcal{X}_t \\ \boldsymbol{u}(t) \in \mathcal{U}_t \\ \boldsymbol{y}(t) \in \mathcal{Y}_t \\ \phi_0(\boldsymbol{x}(t_0), \boldsymbol{u}(t_0), \boldsymbol{y}(t_0), t_0) = 0 \\ \phi_f(\boldsymbol{x}(t_f), \boldsymbol{u}(t_f), \boldsymbol{y}(t_f), t_f) = 0 \end{cases} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^p$, $\boldsymbol{u} \in \mathbb{R}^q$, and $\boldsymbol{y} \in \mathbb{R}^r$. It is assumed that the feasible region consisting of the sets $\mathcal{X}_t$, $\mathcal{U}_t$ and $\mathcal{Y}_t$ can be represented as a finite collection of nonlinear equality and

978-1-5386-1946-9/18/$31.00 ©2018 IEEE

inequality constraints. It is also assumed that the problem has fixed end points, as indicated by the last two equalities. Here, the focus is on time-optimal trajectories, but other objectives are straightforward to include.

B. Timed Elastic Band Representation

Consider the discretization of the state vector and input represented by,

$$\boldsymbol{x} := \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\},$$
$$\boldsymbol{u} := \{\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_{n-1}\},$$

where indices 1 to n represents the time instances, and the finite time difference is given by ΔT. Note that n and ΔT are allowed to vary as the band deforms in time. n will be bounded between some predefined $\bar{n}_{min}$ and $\bar{n}_{max}$, while ΔT must satisfy $\Delta T > 0$. The discrete states and inputs are constrained by approximated system dynamics, obtained through forward differences, $\frac{\boldsymbol{x}_{k+1}-\boldsymbol{x}_k}{\Delta T} = f(\boldsymbol{x}_k, \boldsymbol{u}_k)$. It is assumed that ΔT is kept small such that the equivalent sampling frequency is high enough to have a stable integration. In practice this imposes an additional constraint on $\bar{n}_{min}$ introduced above. Following the notation of [8], the state and input sequences, along with the time difference, are lumped into a so called *Timed Elastic Band* (TEB) set $\mathcal{B} \subseteq \mathrm{R}^d$ where $d = np + (n-1)q + 1$,

$$\mathcal{B} := \{\boldsymbol{x}_1, \boldsymbol{u}_1, \boldsymbol{x}_2, \boldsymbol{u}_2, \dots, \boldsymbol{x}_{n-1}, \boldsymbol{u}_{n-1}, \boldsymbol{x}_n, \Delta T\}.$$

The set constitutes the TEB and can now be deformed in space and time while aiming to satisfy the approximated system dynamics between each state.

The general trajectory planning problem (1) can now be rewritten in the TEB framework as a nonlinear optimization problem:

$$\begin{aligned}
\min_{\mathcal{B}} \quad & (n-1)\Delta T \\
\text{s.t.} \quad & \frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\Delta T} - f(\boldsymbol{x}_k, \boldsymbol{u}_k) = 0 \\
& \boldsymbol{x}_k \in \mathcal{X}_k \\
& \boldsymbol{u}_k \in \mathcal{U}_k \\
& g(\boldsymbol{x}_k, \boldsymbol{u}_k) \in \mathcal{Y}_k \\
& \phi_s(\boldsymbol{x}_1, \boldsymbol{u}_1, g(\boldsymbol{x}_1, \boldsymbol{u}_1)) = 0 \\
& \phi_f(\boldsymbol{x}_n, \boldsymbol{u}_n, g(\boldsymbol{x}_n, \boldsymbol{u}_n)) = 0 \\
& \Delta T > 0, \ k \in [1, n-1]
\end{aligned} \tag{2}$$

for a fixed $\bar{n}_{min} \leq n \leq \bar{n}_{max}$. The above formulation can be simplified further if the inverse relation, $\boldsymbol{x}_k = g^{-1}(\boldsymbol{y}_k, \boldsymbol{u_k})$, is available. In this case the constraints and endpoint equalities in $\boldsymbol{y}_k$ can be written entirely in terms of $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$. Note that due to the discretization scheme each constraint k typically only involves neighboring points, leading to a sparse optimization problem.

C. Trajectory Deformation through TEB based Nonlinear Model Predictive Control

The optimization solution should be acquired fast to achieve real-time performance. Therefore, it is generally not possible to determine a global solution in each trajectory update. Instead, the aim will be to determine a feasible trajectory that can be improved during subsequent iterations by employing NMPC. During each control cycle, the current trajectory is improved through an iterative optimization method, and then only the first state and input is used as control action. In the next cycle, the current state is obtained directly from sensor readings or from state estimation.

The trajectory planner stores the currently planned trajectory as a TEB set $\mathcal{B}^*$. During each control cycle, the trajectory is deformed, first in time, and then in space. The initial trajectory is determined simply as a linear interpolation between start and goal state.

1) Deformation in time: The length of the timed elastic band, $\mathcal{B}^*$, will be varied according to a simple decision rule, as suggested in [8]. Let $\Delta\bar{T}_{ref}$ denote a predefined reference time related to the sampling time and define $\Delta\bar{T}_{hyst} \approx 0.1\Delta\bar{T}_{ref}$ to avoid oscillations during updates. See Algorithm 1 where the full update is outlined. The predicted total time should not change, therefore $\Delta T_{i+1} n_{i+1} = \Delta T_i n_i$. Inserting a new state will decrease ΔT, referred to as *contracting* in time. Similarly, removing a state will increase ΔT, defined as *expanding* in time. During each control cycle, the TEB update is performed a predefined $\bar{I}_{TEB}$ number of times. The complete length of the TEB set $\mathcal{B}^*_{i+1}$ after an update is given by, $\bar{d} = n_{i+1}p + (n_{i+1}-1)q + 1$. Consequently, (2) changes in each iteration. To increase the chance of retaining feasibility, the trajectory is re-interpolated linearly between TEB updates.

2) Deformation in space: After updating the length of the trajectory, some variation of (2) will be solved for the current value of n. Each non-linear optimization problem is solved using Sequential Quadratic Programming (SQP). The SQP Solver runs for a predefined $\bar{I}_{SQP}$ number of iterations after each TEB update. Hence, the complete number of optimization iterations performed each control cycle is given by $\bar{I}_{TEB} \cdot \bar{I}_{SQP}$.

Employing SQP requires derivative evaluations of the objective and the constraints of (2). Here, gradients, Hessians and Jacobians are obtained through automatic differentiation [9]. These methods can also determine sparsity patterns of the Hessians and Jacobians using graph coloring techniques. The sparse information is later utilized in the SQP procedure. Apart from memory savings and efficient matrix-vector operations, efficient sparse factorizations can be utilized when solving the quadratic subproblems. This is an important extension to the work in [8] where the sparsity of the problem was not explored. Without exploring the sparsity the real time performance cannot be achieved.

D. Enforcing Convergence and Stability through Switching Strategies

There is no guarantee that the suboptimal planning approach described here always yields a trajectory that successfully converges to the target state, nor stabilizes the system at the target. Also, numerical instabilities may arise as the target state is approached since ΔT would approach zero.

Algorithm 1 Trajectory Deformation

Input: $\mathcal{B}$ - trajectory as TEB set; $\mathcal{O}$ - environment information, S - control strategy
Output: Improved trajectory $\mathcal{B}$
function TEBSolve($\mathcal{B}$, $\mathcal{O}$, S)
  $\tilde{\mathcal{B}} \leftarrow \mathcal{B}$ ▷ cache previous trajectory
  for each iteration 1 to $\bar{I}_{TEB}$ **do**
    if $\Delta T > \Delta\bar{T}_{ref} + \Delta\bar{T}_{hyst}$ **then**
      $\mathcal{B} \leftarrow$ ContractInTime($\mathcal{B}$)
    else if $\Delta T < \Delta\bar{T}_{ref} - \Delta\bar{T}_{hyst}$ **then**
      $\mathcal{B} \leftarrow$ ExpandInTime($\mathcal{B}$)
    end if ▷ Deform trajectory in time
    $F \leftarrow$ GenerateObjectives(S, $\mathcal{B}$, $\mathcal{O}$)
    $C \leftarrow$ GenerateConstraints(S, $\mathcal{B}$, $\mathcal{O}$)
      ▷ Reformulate underlying optimization problem
    $\mathcal{B} \leftarrow$ SQPSolve($\mathcal{B}$, F, C) ▷ Deform trajectory in space
    if $\mathcal{B}$ not feasible or SQP solver fails **then**
      $\mathcal{B} \leftarrow \tilde{\mathcal{B}}$ ▷ revert to previous feasible trajectory
      return $\mathcal{B}$
    end if
    if SatisfiesFirstOrderKKT($\mathcal{B}$, F, C) **then**
      return $\mathcal{B}$
    end if
  end for
end function

It has been shown that under some mild conditions suboptimal model predictive controllers are stabilizing [10]. In brief, as long as feasibility is maintained, the state is steered towards the terminal state. One of the conditions is that during each iteration there is a sufficient decrease in the objective, which in [10] is ensured by assuming quadratic objective functions.

To utilize these ideas in the approach presented here, a switching strategy is proposed. When the system is sufficiently close to the target, the underlying optimization problem is modified into an optimization problem where

$$\sum_{k=1}^{n-1} ||\boldsymbol{x}_k - \boldsymbol{x}_f||^2$$

is minimized to enforce convergence. Note that the inherent sampling time $\Delta\bar{T}_{sample}$ is used in the tracking formulation. This is to ensure that time deformation does not disrupt the convergence properties of the quadratic formulation. As an option, in a small region, the stability region, a linear LQR controller can be used.

The sizes of the regions where strategy switches occur are application dependent. Simple applications could utilize the quasi time-optimal strategy for the main part of the planning duration. In contrast, harder problems might require more conservative conditions for switching to enforce convergence and stability.

E. Multiple Trajectories

Since the underlying optimization problem is non-convex in general, the solution is typically a local optimum and will depend on the initial trajectory guess. Therefore, multiple starting guesses are supplied to initiate multiple planning procedures in parallel. Since the planning procedures are independent, they can be efficiently planned in parallel as long as the number of trajectories does not exceed the number of available processing elements. The initial starting guesses are here generated through linear interpolation between the initial state and the target state. Multiple end targets are created from solutions of $g^{-1}(\boldsymbol{y}_k, \boldsymbol{u_k})$ in (1). During each control cycle, the planner chooses among the trajectory candidates and applies the corresponding control. Here, the selection rule is simply to choose the trajectory with current best objective value. As one trajectory is traversed, it will become less likely that a switch would occur as the other candidates will become more ill-posed. Hence, the planner can eventually commit to a trajectory when the objective gap becomes large, and drop the other candidates to free up computational resources.

III. Implementation

The complete trajectory optimization procedure is condensed into a real-time algorithm, shown in Algorithm 2. The solver is implemented as an object-oriented software package in C++. The most notable classes are the `TrajectoryPlanner`, `TEBSolver` and `SQPSolver`. The `TrajectoryPlanner` manages the planning procedure and provides API for end users. The `TEBSolver` is responsible for deforming the trajectory, which is internally stored in the data type `TimedElasticBand`. The `SQPSolver` is a general purpose solver for non-linear optimization problems and is used by the `TEBSolver` to improve the trajectory.

The package utilizes several third-party libraries. *Eigen* [11], for matrix/vector storage and linear algebra operations. *Spectra* [12], for large scale eigenvalue problems built upon *Eigen*. *qpOASES* [13], used to solve the quadratic sub-problems that arise during the SQP procedure. *HSL_MA57* [14] used internally in qpOASES to efficiently solve a sparse symmetric linear system. Finally, *CppAD* [15], used to compute derivative information as well as sparsity patterns for Hessians and Jacobians.

IV. Simulation tests

The planning procedure is evaluated in simulations with the parameters $\bar{I}_{TEB} = \bar{I}_{SQP} = 2$ and $\Delta\bar{T}_{sample} = \Delta\bar{T}_{ref} = 0.05$ sec. The simulations use the Julia [16] programming language. Specifically, the libraries [17], [18] and [19] were utilized. All simulations were performed on a quad core Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz processor.

A. Model

A Planar Elbow robot of SCARA type is used in the simulations. The model consists of two links with revolute joint angles, q_1 and q_2. The state variables, $\boldsymbol{x} = (q_1, q_2, \dot{q}_1, \dot{q}_2)^T$. The output, $\boldsymbol{y}$, is the Cartesian coordinates of the end-effector, provided by the forward kinematics $\boldsymbol{y} = \chi(q_1, q_2)$. The dynamical equations of motion are given by,

$$M(q_1, q_2)\begin{pmatrix}\ddot{q}_1\\ \ddot{q}_2\end{pmatrix} + C(\dot{q}_1, \dot{q}_2)\begin{pmatrix}\dot{q}_1\\ \dot{q}_2\end{pmatrix} + \begin{pmatrix}c_1\dot{q}_1\\ c_2\dot{q}_2\end{pmatrix} = \begin{pmatrix}\tau_1\\ \tau_2\end{pmatrix},$$

where the inertia matrix $M(q_1, q_2)$ and the centripetal and Coriolis terms $C(\dot{q}_1, \dot{q}_2)$ can be obtained through analytical mechanics. Linear friction terms c_1 and c_2 are also included in the model and the parameter values are provided in Table I. The right hand side contains the torque vector $\boldsymbol{\tau}(t)$ which is used as the control variable. See for example [20] for a thorough introduction to the Planar Elbow model.

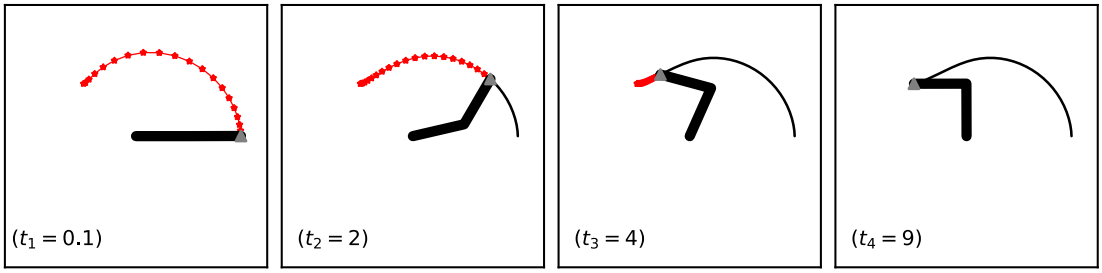
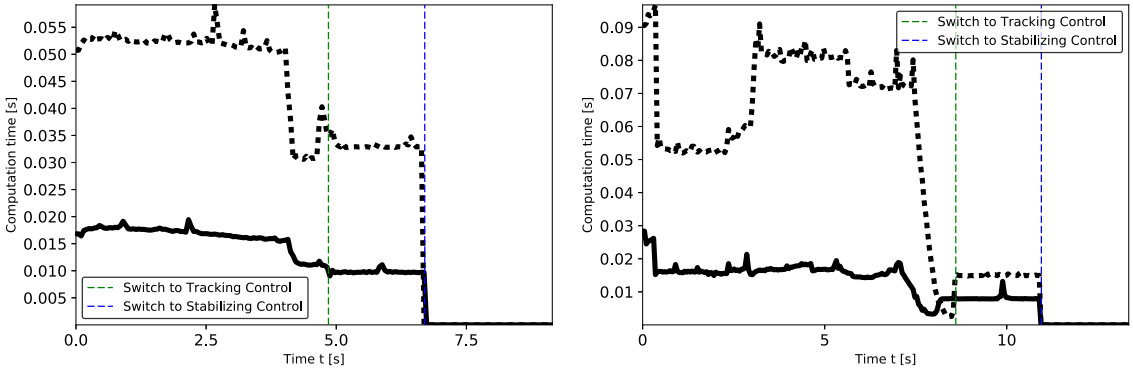


Fig. 1: The gray triangle represents the end-effector, which is moved to the target location. Red path is planned path in each time frame. The realized trajectory is shown in black.



(a) Planning scenario shown in Fig. 1.

(b) Planning scenario shown in Fig. 5.

Fig. 2: Computation time required to recompute the trajectory during the simulated planning procedures. The solid line in the respective figure is when sparse information is exploited and the dotted line corresponds to treating the problem as dense.

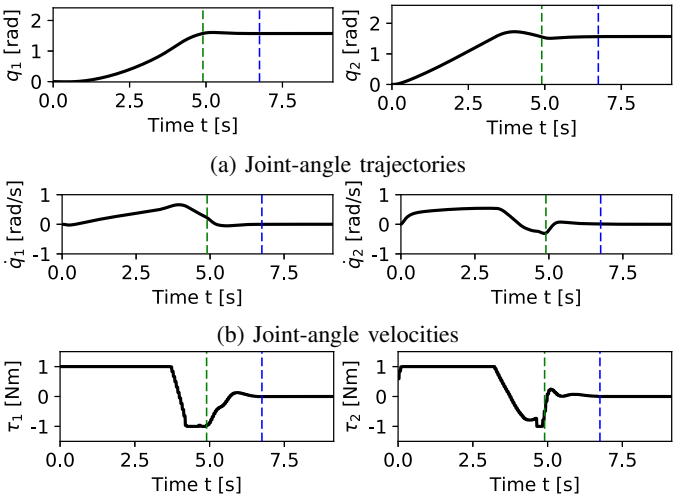
TABLE I: parameter values for the *PlanarElbow* model.

m_1	1kg	c_1	1.5 Nms/rad
m_2	1kg	c_2	1.5 Nms/rad
l_1	1m	$\lvert q_1 \rvert_{max}$	2π
l_2	1m	$\lvert q_2 \rvert_{max}$	π
I_1	0.5kgm^2	$\lvert \dot{q} \rvert_{max}$	2 rad/s
l_2	0.5kgm^2	$\lvert \tau \rvert_{max}$	1 Nm

B. Simple Motion

In a first simple example, the task is to move the end-effector from start-point to end-point. Here, The radius of the tracking vicinity is set to 20 mm and the radius of the stability vicinity is set to 5 mm. The resulting planning procedure and the realized trajectory is shown in Fig. 1. The state (joint angles and joint velocities) trajectories and the applied control signals are shown in Fig. 3. The control signals are of bang-bang nature, which is indicative of quasi time-optimality, until the switch to tracking occurs. The combination of the tracking procedure and stabilizing control successfully brings the end-effector to a halt at the target location. The second joint angle has a larger overshoot and a less smooth control signal, which is expected as the cross couplings make the second arm harder to control. Measures of the required computation time is shown in Fig. 2a.

During the main planning procedure, the trajectory length and hence the size of the underlying optimization problem is allowed to vary. Consequently, the computation time required to replan the trajectory varies between control cycles. After switching to tracking control, the problem size is fixed and



(a) Joint-angle trajectories

(b) Joint-angle velocities

(c) Control inputs

Fig. 3: Trajectory planning procedure of the SCARA robot arm. Dashed vertical lines indicate when the control strategy switches from Quasi time-optimal to Tracking, and from Tracking to Stabilizing LQR control.

Algorithm 2 Trajectory Planning

Input: $\boldsymbol{q}_s$ - current state; $\dot{\boldsymbol{q}}_s$ - current velocity; $\boldsymbol{y}_f$ - target; $\dot{\boldsymbol{y}}_f$ - target velocity; $\mathcal{O}$ - obstacle information; S - control strategy
Output: (Sub-)optimal control input $\boldsymbol{u}$

procedure PLANTRAJECTORY(S)
  repeat
    $(\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s, \boldsymbol{y}_f, \dot{\boldsymbol{y}}_f) \leftarrow$ READCURRENTSTATE
    if not initialized or new far-off target **then**
      SWITCHSTRATEGY(S)
      $T \leftarrow$ allocate collection of trajectories
      if handle multiple trajectories **then**
        for each possible goal state $\boldsymbol{q}_f$ **do**
          $\mathcal{B} \leftarrow$ INITIALIZETRAJECTORY($\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s, \boldsymbol{q}_f, \dot{\boldsymbol{q}}_f$)
          T $\leftarrow$ APPENDTRAJECTORY(T, $\mathcal{B}$)
        end for
      else ▷ Only one maintained trajectory
        $\boldsymbol{q}_f \leftarrow$ DETERMINECLOSESTGOALSTATE
        $\mathcal{B} \leftarrow$ INITIALIZETRAJECTORY($\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s, \boldsymbol{q}_f, \dot{\boldsymbol{q}}_f$)
        T $\leftarrow$ APPENDTRAJECTORY(T, $\mathcal{B}$)
      end if
    end if
    if within tracking vicinity of target **then**
      SWITCHSTRATEGY(Track)
    end if
    if within close vicinity of target **then**
      SWITCHSTRATEGY(Stabilize)
    end if
    if strategy is Stabilize **then**
      $\boldsymbol{u} \leftarrow$ LQR($\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s, \boldsymbol{q}_f, \dot{\boldsymbol{q}}_f$)
    else
      $\mathcal{O} \leftarrow$ READSENSORINPUT ▷ Environment information, such as obstacles
      for each maintained trajectory $\mathcal{B} \in T$ **do** ▷ on separate threads
        $\mathcal{B} \leftarrow$ UPDATETRAJECTORY($\boldsymbol{q}_s, \dot{\boldsymbol{q}}_s, \boldsymbol{y}_f, \dot{\boldsymbol{y}}_f$) ▷ shift-initilization
        $\mathcal{B} \leftarrow$ TEBSOLVE($\mathcal{B}, \mathcal{O}, S$) ▷ improve the trajectory
      end for
      $\mathcal{B}^* \leftarrow$ SELECTBESTTRAJECTORY(T)
      $\boldsymbol{u} \leftarrow$APPLYCONTROL($\mathcal{B}^*$)
    end if
  until target has been reached
end procedure

the computation time does not vary. During the final phase of stabilizing control, the planner uses the already computed LQR feedback and the required time is barely noticable. By exploiting the sparsity in the problem the computation time is reduced by a factor 3. Notably, the required computation time never exceeds the sampling time, which indicates that the procedure could be viable for real-time control. In a real application the free resources during the last two phases can be used to initiate the planning of the next motion to find a good initial trajectory to start the next NMPC solution.

C. Motion in Environment with Obstacles

Now, three obstacles are introduced to make the planning problem more complex. For simplicity, the robot arm is regarded as being placed at a level above the obstacles, safe from collision. However, a tool fixed at the end-effector reaches down to the lower level and can collide with the obstacles. Furthermore, the obstacles are modeled as circular areas that the end-effector cannot cross.

Obstacle avoidance can be included in the planning framework by introducing penalizing terms in the objective for every point on the planned trajectory sufficiently close to any of the obstacles. At a closer distance, hard constraints are used to ensure that no collision occurs. These distances also make up configuration parameters that are input to the planner. In these examples, the radius of the stability vicinity is set to 0.1 mm.

The resulting planning procedure and the realized trajectory is shown in Fig. 4. Here, the planner does not succeed in deforming the initial guess into a collision-free path. The planner might succeed if the configuration parameters were tweaked slightly. However, the approach here will be to employ the multiple trajectory procedure described in Section II-E. If the planner is instead supplied with 4 different starting guesses, the resulting planning procedure and the realized trajectory is shown in Fig. 5.

Now, as the planner can explore 4 different paths simultaneously, it successfully determines a collision-free trajectory. Note that the planner commits to a trajectory already after 0.4 s. The required computation time is shown in Fig. 2b. Notably, the overhead from choosing among the trajectory candidates is only apparent in the initial iterations. After committing to a trajectory the solution times are comparable to the simple scenario.

V. CONCLUSION AND FUTURE WORK

A general trajectory planning problem is addressed. The procedure presented here is able to reshape the path in response to sensor inputs and obstacle avoidance. The path is adopted to the environment by allowing the underlying optimization formulation to change each control cycle. The presented approach relies on different stages. The first stage involves the TEB based NMPC approach, which yields quasi time-optimal solutions. The second stage employs a tracking formulation where the goal becomes to minimize the distance to the target. Finally, in an optional last stage, a feedback controller stabilizes the system using LQR. A number of different scenarios related to robot trajectory planning and control are provided to motivate the use but the possible applications are not limited to robotic manipulators. In simulation it is shown that real-time performance can be achieved for a realistic robot model. The key to achieve performance is to exploit the sparsity in the problem which in this case provides a reduction of the computation time by a factor 3.

There are several directions for future work. The real-time capability could be evaluated further by employing the presented procedure in an embedded setting, controlling a real robot. Another idea is to investigate alternative solution approaches to the underlying non-linear optimization problems. For example, a trust-region based SQP approach is introduced in [21], where the optimization iterates are guaranteed to be feasible through a perturbation process. Another direction could be to investigate more sophisticated TEB updates. If obstacles are present, the planner could add more states, not just one each cycle, to change the trajectory resolution to a suitable level faster. Likewise, if the path is clear, states could be removed at a faster rate to reach a crude resolution which is sufficient for planning the remainder of the trajectory. The initialization of the trajectory could also be improved by using a more sophisticated guess. One idea could be to first adopt the efficient two-step approach to generate an initial feasible trajectory. Initial collision free paths could be acquired by employing a probabilistic roadmap method [22]

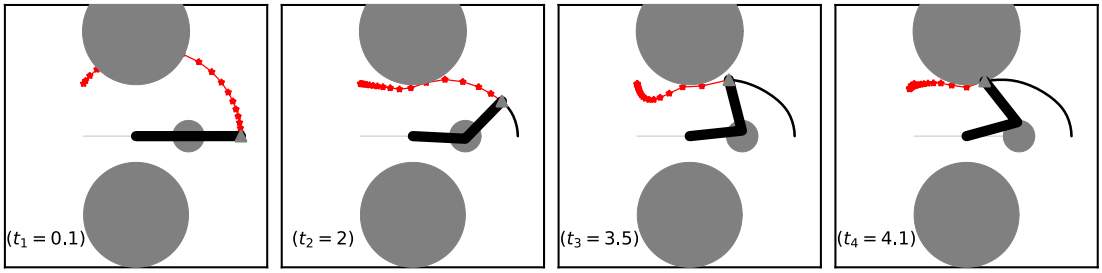


Fig. 4: Result from example with 3 circular obstacles. The gray triangle represents the end-effector, which is moved towards the target location but collides with one of the obstacles. The planned trajectory is red and the realized trajectory is black.

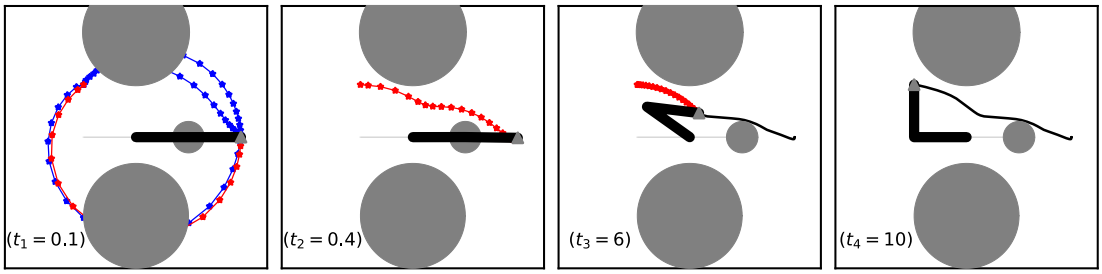


Fig. 5: Result from example with 3 circular obstacles when planning multiple trajectories. The gray triangle represents the end-effector, which is moved towards the target location without colliding with the obstacles. The planned trajectory is shown in ref and the realized trajectory is shown in black. Alternative trajectories that are planned in parallel are shown in blue.

in conjunction with handling multiple trajectories, akin to the approach presented in [7].

VI. Acknowledgment

This work was supported by the VINNOVA industrial excelence center LINK-SIC at Linköping University and the KTH initiative, digitalization of Swedish industry.

References

[1] C.-S. Lin, P.-R. Chang, and J. Y. S. Luh, "Formulation and optimization of cubic polynomial joint trajectories for industrial robots," *IEEE Trans. Autom. Control*, vol. 28, no. 12, pp. 1066–1074, Dec. 1983.

[2] O. von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals Oper. Res.*, vol. 37, pp. 357–373, 1992.

[3] K. G. Shin and N. D. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Trans. Autom. Control*, vol. 30, no. 6, pp. 531–541, 1985.

[4] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *Automatic Control, IEEE Transactions on*, vol. 54, no. 10, pp. 2318 –2327, oct. 2009.

[5] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 802–807.

[6] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. VDE, 2012, pp. 1–6.

[7] C. Rosmann, F. Hoffmann, and T. Bertram, "Planning of multiple robot trajectories in distinctive topologies," in *Mobile Robots (ECMR), 2015 European Conference on*. IEEE, 2015, pp. 1–6.

[8] ——, "Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control," in *Control Conference (ECC), 2015 European*. IEEE, 2015, pp. 3352–3357.

[9] A. Griewank and A. Walther, *Evaluating Derivatives*, 2nd ed. Society for Industrial and Applied Mathematics, 2008. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9780898717761

[10] P. O. M. Scokaert, D. Q. Mayne, and J. B. Rawlings, "Suboptimal model predictive control (feasibility implies stability)," *IEEE Transactions on Automatic Control*, vol. 44, no. 3, pp. 648–654, Mar 1999.

[11] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[12] Y. Qiu, "Spectra v0.5," http://yixuan.cos.name/spectra/.

[13] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[14] "The HSL mathematical software library: HSL ma57sparse symmetric system: multifrontal method." http://www.hsl.rl.ac.uk/catalogue/hsl_ma57.html, accessed: 2017-02-01.

[15] Bradley M. Bell, Coin-Or Foundation, "Cppad." [Online]. Available: http://www.coin-or.org/CppAD/

[16] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *CoRR*, vol. abs/1411.1607, 2014. [Online]. Available: http://arxiv.org/abs/1411.1607

[17] "Julia library for solving ordinary differential equations," https://github.com/JuliaDiffEq/OrdinaryDiffEq.jl, accessed: 2017-01-15.

[18] "Julia library for wrapping c++ code," https://github.com/JuliaInterop/CxxWrap.jl, accessed: 2017-01-15.

[19] "Julia library for plotting," https://github.com/JuliaPlots/Plots.jl, accessed: 2017-01-15.

[20] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.se/books?id=wGapQAAACAAJ

[21] M. J. Tenny, S. J. Wright, and J. B. Rawlings, "Nonlinear model predictive control via feasibility-perturbed sequential quadratic programming," *Computational Optimization and Applications*, vol. 28, no. 1, pp. 87–121, 2004.

[22] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug 1996.