2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Trajectory Planning of Slider-Pushers in Cluttered Environments with Automatic Switching

Thomas Neve*, Sander De Witte, Tom Lefebvre and Guillaume Crevecoeur

***Abstract*—Non-prehensile manipulation presents a paradigm for manipulating objects that aims to extend the versatility of robotic tasks beyond conventional grasping. Integrating motion primitives like pushing, tipping, rolling, throwing, and sliding into robots brings forth a set of challenges related to sensing, control, and planning. In particular, planning becomes notably intricate when dealing with densely arranged obstacles. This work focuses on the act of pushing an object through a series of obstacles within a cluttered environment. In such a setting, it may become necessary to change the direction of the pushing action in order to navigate through narrow passages. Incorporating these directional switches introduces a symbolic level of decision making casting the problem in the task and motion planning framework. In this work we introduce a novel optimization-based algorithm tailored to a pusher-slider system, capable of handling highly constrained regions. This algorithm automatically determines a switching sequence and the corresponding trajectories. The infinite motion optimization problem is transcribed to a nonlinear program using B-splines. Our algorithm solves the problem jointly optimizing switching and trajectory parameters to establish a feasible path through the environment. Directional switches are introduced based on the properties of the intermediate solution. The work is demonstrated through simulation experiments in various cluttered environments.**

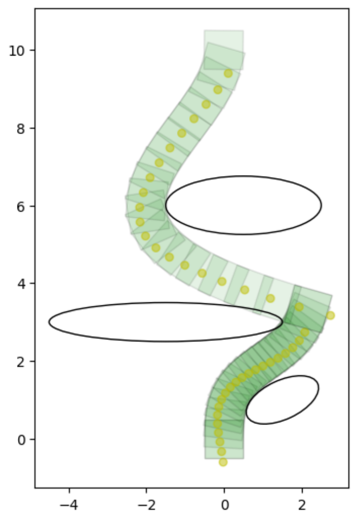


Fig. 1: *Trajectory of a pusher-slider system in a constrained environment which requires a switch in pushing direction.*

I. Introduction

When it comes to interacting with objects in our daily lives, our actions go beyond mere grasping and releasing. We engage in a diverse range of tasks, including pushing, pulling, sliding, and lifting. This allows us to effectively manipulate objects of many shapes, sizes and weights. However, most practical applications of robotics do not reach that level of non-prehensile manipulation and are limited to some form of pick-and-place operations. This is driven by the fact that once an object is grasped, reliable control is more easily achieved without the need for continuous sensing and complex interactions. By widening the array of manipulation techniques, new avenues for tackling tasks opens up in a variety of fields [1]–[3].

In this work, we focus on non-prehensile manipulation, pushing in particular. While this aspect presents a number of challenges when it comes to planning and control, it empowers a robotic manipulator to handle objects that may be too large or densely arranged to be feasibly grasped [4], [5]. Furthermore, we argue that pushing an object allows for a streamlined end-effector design without compromising the object's maneuverability significantly. Here we are interested in the control of a sliding object in the presence of densely arranged obstacles.

The authors are with the Dynamic Design Lab (D^{2}LAB) of the Department of Electromechanical, Systems and Metal Engineering, Ghent University, B-9052 Ghent, Belgium.
The authors are member of core-lab MIRO, Flanders Make, Belgium.
*Corresponding author: `thomas.neve@ugent.be`.

The pusher-slider system exemplifies a fundamental non-prehensile manipulation task. The objective is to control the motion of the slider via a single point of contact, the pusher. In a densely cluttered environment keeping contact with the pusher quickly becomes infeasible and a switch in pushing direction is often required, Fig. 1. Such a directional switch severely complicates the controller design.

We can formulate the problem as a short-horizon task and motion planning (TAMP) problem [6], where the task level dictates the switching sequence, and the motion planning determines the trajectory in between two switches. TAMP addresses a combined problem involving a finite sequence of discrete mode types, continuous mode parameters, and the continuous motion paths. In standard TAMP approaches these parameters are determined in an integrated way, effectively managing the interdependencies between motion-level and task-level elements of the problem. Such a problem could be solved by introducing a set of discrete variables, each representing a discrete mode of operation. This results in a large mixed-integer programming (MIP) problem [7] which itself is computationally expensive to solve.

Under the assumption of frictionless slider-pusher contact, it can be demonstrated that the pusher-slider system is differentially flat [8]. Since the pusher-slider exhibits the differential flatness property, the set of optimization variables for the trajectory optimization can be reduced to a subset of states, from which the required control actions can be computed retroactively using the flat expressions of the system. This property can be particularly useful for both solving trajectory planning [9]–[12], and tracking problems [13]–[15]. The work [10]–[12] demonstrated the application of inversion based trajectory optimization to plan trajectories in an output space. Other work employs inverse dynamics with a generic time-elastic-band formulation for car-like robots to plan continuous trajectories in the state space [16].

In the context of a pusher-slider system, non-prehensile manipulation planning has been tackled through gradient based trajectory optimization [17], [18] and sample based planners [19], [20]. In [17] a hybrid approach is proposed that incorporates the idea of switching between different pushing faces, considering a series of potential switching sequences. The downside is the significant cost for determining a feasible, and optimal, switching sequence amongst the potential set. In [18] different contact modes with the slider were considered through trajectory optimization with complementary constraints, though discrete switching between the slider's faces was not considered. Both in [20] and [19] a different approach is taken, exploring the state space through a sample based planner. Navigation, including switching from pushing face, can then be handled through Dijkstra, or with the concept of feasible sets respectively. While these methods can be suitable for discovering trajectories through cluttered environments, their computational complexity can be prohibitive. To that end, we propose a novel gradient-based optimization algorithm, tailored to the pusher-slider system, which can incorporate a switch in pushing mode. We employ a B-spline parameterization of the flat trajectory to leverage the differential flatness properties of the pusher-slider system. In previous work this has proven beneficial with respect to a shooting approach [21]. By transcribing the subsequent trajectories using B-splines, we can effectively incorporate the discrete switches in a numerical optimization problem. Multiple B-splines are chained together using equality constraints on a subset of the end states of the B-spline. The orientation of the slider is constrained in such a way so that switching from pushing direction becomes feasible, allowing for a switch to occur inbetween B-spline connections. We can then tackle the TAMP problem by iteratively determining a switching sequence and solving the complete trajectory optimization problem, with the switches included. Additionally, a way of finding the switching sequences, tailored to the pusher-slider system, is proposed. By detecting high local curvature of the trajectory we determine where a switch in pushing direction is beneficial for the overall feasibility of the pusher-slider trajectory. Results show that the proposed methodology can autonomously determine a feasible switching sequence and path through a set of densely arranged obstacles.

II. Quasi-static Slider-Pusher Model

A. Kinematics

The pusher-slider system consists of a sliding object (the slider) and a single contact point (the pusher). It is assumed in the remainder of this work that contact is established at all times. If this is the case we can model the system using the planar configuration of the (square) object determined by the coordinates, x and y, and orientation, ϕ. The location of the pusher is determined by storing the pushing face, s, and the distance of the contact point with respect to the center of that face, c. These are combined in a state vector, $\mathbf{x} \in \mathbb{R}^4$ where $\mathbf{x} = (x, y, c, \phi)$. The pushing face variable, s, is discrete and is therefore not included in the state vector. We further assume that an arbitrary planar velocity can be imposed to the pusher. The inputs v_t and v_n denote, respectively, the speed tangent and normal to the surface against which is pushed and are therefore defined in a local frame of reference. The velocities are combined in the input vector, $\mathbf{u} \in \mathbb{R}^2$.

Provided that the inertial forces are negligible compared with the friction force between the sliding object and the supporting surface the interaction becomes quasi-static. In this work we additionally assume that the interaction between the pusher and the slider is frictionless.

If both assumptions are valid, we can derive the following differential kinematic model [8].

$$\begin{aligned}
\dot{x} &= -\tfrac{\beta^2}{\beta^2+c^2} v_n \sin(\phi) \\
\dot{y} &= \tfrac{\beta^2}{\beta^2+c^2} v_n \cos(\phi) \\
\dot{c} &= v_t - \left(\tfrac{b}{2} + r\right) \tfrac{c}{\beta^2+c^2} v_n \\
\dot{\phi} &= \tfrac{c}{\beta^2+c^2} v_n
\end{aligned} \tag{1}$$

Here β^2 is a geometric factor that depends on the dimensions of the slider. In the case of a rectangular geometry, $\beta^2 = \frac{1}{12}(a^2 + b^2)$ where the parameters a and b are geometric with a and b denoting the length of the slider's side. Further, the radius of the pusher is denoted as r.

The equation above describes the dynamics of the system for a given fixed face variable, s. When arbitrary changes in the pushing face are allowed, the dynamics are extended to a hybrid formulation. Assuming a switch from s to s' at time t, we have that $\mathbf{x}(t^-) = \mathbf{x}(t^+)$ with the exception of $c(t^+)$ which can be chosen arbitrarily. The same applies for the input, $\mathbf{u}(t^+)$. From time t^+ onwards the dynamics are described by (1) rotated over 90 or -90 degrees.

B. Differential flatness

Differential flatness, or simply flatness, extends the notion of an inverse dynamics function to a restricted class of underactuated nonlinear systems. Differential flatness implies the existence of a set of differentially independent variables, $\boldsymbol{\xi}$, the so-called flat output. Any feasible state-input trajectory, i.e. any combination $(\mathbf{x}, \mathbf{u})$ that satisfies (1) can be expressed as a function of $\boldsymbol{\xi}$ and its higher order time derivatives.

Under the assumption of frictionless slider-pusher contact, it can be demonstrated that the equations of motion (1) are differentially flat [8]. Specifically, for the pusher-slider system, the Cartesian coordinates serve as the flat output, $\boldsymbol{\xi} = (x, y)$. Then the control input and auxiliary states can be expressed as

$$\begin{aligned}
c &= \beta^2 \tfrac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\sqrt{\dot{x}^2+\dot{y}^2}^3} \\
\phi &= -\arctan \tfrac{\dot{x}}{\dot{y}} \\
v_t &= \beta^2 \tfrac{\dot{x}\dddot{y} - \dddot{x}\dot{y}}{\sqrt{\dot{x}^2+\dot{y}^2}^3} + 3\beta^2 \tfrac{(\ddot{x}\dot{y} - \dot{x}\ddot{y})(\dot{x}\ddot{x}+\dot{y}\ddot{y})}{\sqrt{\dot{x}^2+\dot{y}^2}^5} + \left(\tfrac{b}{2} + r\right) \tfrac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\dot{x}^2+\dot{y}^2} \\
v_n &= \left(1 + \beta^2 \tfrac{(\ddot{x}\dot{y} - \dot{x}\ddot{y})^2}{(\dot{x}^2+\dot{y}^2)^3}\right) \sqrt{\dot{x}^2 + \dot{y}^2}
\end{aligned} \tag{2}$$

Differential flatness is an appealing property for trajectory optimization, as will be illustrated next.

III. Trajectory Optimization with Automatic Switching Logic

In this section we first give a formal introduction to the task and motion planning (TAMP) framework which allows for a generalised formulation of our problem. We continue this section with a description of B-splines, highlighting their relevance in combination with a flat system to represent smooth trajectories in between switches. Following this, we present our switching algorithm which automatically determines if and where a switch in pushing direction of the pusher-slider could potentially benefit the manoeuvring. Finally, we conclude with a description of the transcribed smooth trajectory optimization problem.

A. TAMP problem formulation

Our main goal is to determine an optimal feasible path from initial state $\mathbf{x}_0$ to goal state $\mathbf{x}_T$ and corresponding input, in an obstacle rich environment. In such a cluttered environment, a switch between different pushing faces might be required e.g. to escape a narrow passage. It is implied that besides smooth path variables we also need to determine a sequence of pushing faces and associated switches between subsequent faces. Following the switching dynamics as described in section II, this will introduce discontinuities in the otherwise smooth path.

Formally, this problem can be formulated as a task and motion planning (TAMP) problem [6]. We optimize for the piecewise smooth functions, $\mathbf{x}$, and, $\mathbf{u}$, related to the continuous motion of the system. The framework extends on traditional trajectory optimization by appending the problem with a discrete logic governing the task level. Here both s_m and a_m represent discrete variables. In the context of TAMP, the sequence of actions, $a_{1:M}$, is referred to as the *skeleton*. The decision, a_m, determines the switch from one mode of operation, s_{m-1}, to the next mode of operation, s_m. The feasible transitions between modes of operation is governed by a first order logic, succ. Further we have functions, $\mathbf{h}_{\text{path}}$ and $\mathbf{g}_{\text{path}}$, and, $\mathbf{h}_{\text{switch}}$ and $\mathbf{g}_{\text{switch}}$, that denote the equality and inequality constraints that are active during a mode of operation and during a switch, respectively.

The TAMP problem is then defined as

$$\min_{\mathbf{x},\mathbf{u},s_{0:M},a_{1:M},t_{1:M},M} \int_0^T l(\mathbf{x}(t),\mathbf{u}(t))dt$$
$$\text{s.t.} \begin{cases} \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{x}(T) = \mathbf{x}_T \\ \quad 0 = \mathbf{h}_{\text{path}}(\mathbf{x}(t),\mathbf{u}(t),s_{m(t)}), & \forall t \in [0,T] \\ \quad 0 \geq \mathbf{g}_{\text{path}}(\mathbf{x}(t),\mathbf{u}(t),s_{m(t)}), & \forall t \in [0,T] \\ \quad 0 = \mathbf{h}_{\text{switch}}(\mathbf{x}(t_m),\mathbf{u}(t),a_m,s_{m-1}), & \forall m \in \mathcal{M} \\ \quad 0 \geq \mathbf{g}_{\text{switch}}(\mathbf{x}(t_m),\mathbf{u}(t),a_m,s_{m-1}), & \forall m \in \mathcal{M} \\ s_m \in \texttt{succ}(s_{m-1},a_m), & \forall m \in \mathcal{M} \end{cases} \tag{3}$$

With M the total number of switches and $\mathcal{M} = \{1,\dots,M\}$. The sequence, $t_{1:M}$ determines the switching time stamps and the function, $m(t)$, is a time dependent representation of the mode index.

The resulting problem poses a large mixed-integer program. Given a specific skeleton, or switching sequence $a_{1:M}$, the optimization problem becomes differentiable since $(\mathbf{h},\mathbf{g})_{\text{path}}$ and $(\mathbf{h},\mathbf{g})_{\text{switch}}$ are smooth. The problem can thus be decomposed into two optimization problems. In an inner-loop, a trajectory optimization problem is solved with $M+1$ smooth trajectory pieces. In an outer-loop an integer programming (IP) problem is solved [22]. The IP problem can be solved using an of the shelf branch-and-bound algorithm. In this manner an exact local optimum can be found.

However, though generic solution methods have been proposed in the literature, the computational burden still proves prohibitive in time critical applications. In the present work we propose an alternative solution approach tailored to slider-pusher systems. Our approach starts from a simple trajectory optimization problem and progressively improves the complexity based on an automatic switching logic.

In the present work the modes, s_m, correspond with the pushing face. The switches correspond with allowed subsequent faces. The differential kinematics (1) and obstacles in the environment can be encoded by the path constraints, $(\mathbf{h},\mathbf{g})_{\text{path}}$. The discontinuous switching dynamic can be encoded in the switch constraint, $(\mathbf{h},\mathbf{g})_{\text{switch}}$. We refer to section III-D for further details.

B. B-spline parameterization of smooth motions

The flat expressions (2) of the system allow for the full state and controls to be encoded by a sufficiently smooth flat path. This approach enables the transcription of the trajectory optimization problem into a numerical optimization problem, eliminating the necessity for enforcing dynamic continuity constraints such as are usually included in $\mathbf{h}_{\text{path}}$.

To parameterize the flat output, we consider B-splines. B-splines, first introduced in [23], consist of a union of local curve segments which are each active on a specific interval. Its segmented nature allows for very efficient tailoring to desired local changes [24]. This property also makes it particularly useful for trajectory optimization as it allows for local adjustments to the path without impacting the global behavior, resulting in sparse Jacobian and Hessian structures within the nonlinear programming framework (see section III-D).

The B-spline, $b(\tau)$, is defined as follows

$$b(\tau) = \sum_{i=0}^{n} p_i b_{i,d}(\tau), \quad \tau \in [0,1] \tag{4}$$

The spline is defined over the closed interval $[0,1]$. The set $\mathbf{p} = \{p_0,\dots,p_n\}$ constitutes the set of control points which acts as a set of weights on the basis functions $b_{i,d}$ where d is the order of the B-spline and $n+1$ is the number of control points.

The basis functions can be determined according to the following recursion formula [25].

$$b_{i,0}(\tau) = \begin{cases} 1, & \tau_i \leq \tau \leq \tau_{i+1} \\ 0, & \text{otherwise} \end{cases}$$
$$b_{i,d}(\tau) = \frac{\tau - \tau_i}{\tau_{i+d} - \tau_i} b_{i,d-1}(\tau) + \frac{\tau_{i+d+1} - \tau}{\tau_{i+d+1} - \tau_{i+1}} b_{i+1,d-1}(\tau) \tag{5}$$

It can be seen here that the closed interval $[0,1]$ is determined by a set of $m+1$ knots with $m = d+n+1$ sub intervals.

$$\boldsymbol{\tau} = \{\tau_0,\tau_1,\dots,\tau_m\} \tag{6}$$

At the knot points, the polynomials are joined and connected in a continuous manner. From the recursion (5) it is clear that each basis function $b_{i,d}$, with weight p_i, is only active on a subset of the interval $[0,1]$. More specifically, it is nonzero on the interval $[\tau_i, \tau_{i+d+1})$. This also results in the B-spline not being defined at the start and the ending of the interval $[0,1]$. However, choosing the knot vector with duplicate knots at the ends resolves this issue and also clamps the start and endpoint of the spline to the two end control points, $b(0) = p_0$ and $b(1) = p_n$. This is often referred to as a clamped uniform B-spline.

$$\boldsymbol{\tau} = \{\underbrace{0,\ldots,0}_{d}, \underbrace{0,\ldots,1}_{\text{internal knots}}, \underbrace{1,\ldots,1}_{d}\} \tag{7}$$

In this work we will parameterize the flat output, $\boldsymbol{\xi}$, using a B-spline and control points, $\mathbf{p}$.

$$\boldsymbol{\xi}(\tau; \mathbf{p}_{0:n}) = \sum_{i=0}^{n} \mathbf{p}_i b_{i,d}(\tau), \quad \tau \in [0,1] \tag{8}$$

Remark that since the flat coordinate of the pusher-slider system $\boldsymbol{\xi} = (x, y)$ is two-dimensional, two B-splines are needed to represent the flat trajectory. Put differently, each control point, $\mathbf{p}_i \in \mathbb{R}^2$. Then following the expressions provided in (2), it is possible to deduce the state and action from the trajectory of the flat output. One verifies that these expressions depend on the third order derivative of the flat output at most. Further note that a B-spline function of order d consists of polynomials of order $d-1$. To ensure that the motion is smooth, the B-spline used to represent the flat trajectory should be at least of order $d = 4$. This is usually chosen as the minimum value to avoid numerical issues [26]. The fact that a relatively small amount of control points, $\mathbf{p}_{0:n} \in \mathbb{R}^{n\times 2}$, is needed to fully parameterize a smooth trajectory will become useful later on in the optimization.

C. Automatic Switching Logic

As discussed in section III-A, once a skeleton, $a_{1:M}$, is provided, problem (3) can be transformed into a smooth trajectory optimization problem. Alternative to the use of a branch-and-bound technique, in the present paper we propose a novel algorithm to progressively determine the switching sequence. The overall algorithm is presented in Algorithm 1.

Given a problem set-up, say E, existing of a set of obstacles, a given initial state, and a desired goal state, the algorithm proceeds as follows. The entire slider's path is constructed by concatenating $M+1$ B-splines. For each B-spline we use a fixed hyperparameterization, i.e. the number of knots and the degree is predetermined. These $M+1$ B-splines correspond with $M+1$ sets of control points denoted further as $\mathbf{P}_{0:M} \in \mathbb{R}^{M+1\times n\times 2}$ (see section III-D for an exact definition). At the concatenation between these B-splines a possible switch may occur. The total number of concatenated B-splines and thus number of potential switches is determined iteratively.

The algorithm is initialized by solving the trajectory optimization using a single B-spline, i.e. $M = 0$. We use two criteria to determine whether M should be increased. Either the solver fails or the maximum geometric path curvature is exceeded. Then depending on the criteria that triggered the next iteration of the algorithm, a different initialization procedure is used to start the next optimization routine.

These two cases indicate different failure modes, requiring distinct actions. Firstly, if the path is infeasible, this may simply indicate that the default knot number was insufficient to solve the problem. Increasing M then simply increases the expressiveness of the solution. The number M is increased by 1 and the problem is reinitialized with $M+1$ B-splines. The new set of control points is then $\mathbf{P}_{0:M+1}$. Secondly, whenever the path exceeds a certain curvature we initialize the control points surrounding the high curvature section in such way to elicit a switch by the NLP solver. Based on the previous solution, we have two options. Either reinitialize with the current number of splines or increase the number of splines by one, i.e. to $M+1$. We perform a check to determine if there are any connected splines that do not utilize the potential switch in between. In such cases, the problem is reinitialized with the same number of splines.

It is possible to compute the geometric curvature of the solution to determine exceeding path curvature. However, notice that the flat expression for c in (2) is proportional to the local curvature of a parametric path. In this work we will infer instances of high curvature by examining the values of c within a solution, setting $c_{max} = 0.4$ as the threshold.

The initialization method of the control points $\mathbf{P}^{\text{init}}_{0:M}$, either with linear interpolation or an inverse calculation, will be explained in detail in section III-E.

D. Continous nonlinear program

Given an action skeleton, $a_{1:M}$, the TAMP problem in (3) reduces to a smooth trajectory optimization problem which we further transcribe into a nonlinear program (NLP) using the B-spline parameterization. The optimizer then effectively optimizes a finite set of interconnected splines where a switch in pushing direction can occur in between. The NLP is parameterized by the hyper set of control points, $\mathbf{P}_{0:M}$. Remark that this set exists of sets of control points, $\mathbf{p}_{0:n}$. To distinguish between the control point for each B-spline we will write $\mathbf{P}_m = \mathbf{p}_{m,0:n}$ with m referring to the spline.

$$\mathbf{P}_{0:M} = \{\mathbf{P}_0, \mathbf{P}_2, \ldots, \mathbf{P}_M\} \tag{9}$$

Algorithm 1 Automatic Switching Logic

```
input problem set-up $E$
output $M$, $\mathbf{P}^*_{0:M}$
$M \leftarrow 0$
$\mathbf{P}^{\text{init}}_{0:M}$ = (linear interpolation III-E1)
while not converged do
    $\mathbf{P}^*_{0:M} \leftarrow$ SolveNLP($\mathbf{P}^{\text{init}}_{0:M}$) {III-D}
    if infeasible then
        M ← M+1
        $\mathbf{P}^{\text{init}}_{0:M}$ = (linear interpolation III-E1)
    else if curvature > curvature_max then
        if all switches active then
            M ← M+1
        end if
        $\mathbf{P}^{\text{init}}_{0:M}$ = (inverse calculation III-E2)
    else
        Solution has converged
    end if
end while
```

To get rid of the continuous nature of problem (3), we introduce a collocation set of $K+1$ equidistant points, say $\mathbf{t} = \{t_0, \ldots, t_K\}$ so that $t_0 = 0$, $t_K = 1$ and $t_k < t_{k-1}$. In total this results in $(K+1)\cdot(M+1)$ collocation points for the whole optimization problem. By using the flat expressions (2), we can evaluate the corresponding state, $\mathbf{x}(t_k; \mathbf{P}_m)$, and input, $\mathbf{u}(t_k; \mathbf{P}_m)$ at every spline, m, and every collocation point, t_k. These collocation points are used to approximate the objective function and to evaluate the continuous path inequality constraints, $\mathbf{g}_{\text{path}}$. Then the resulting NLP can be formulated as follows

$$\min_{\mathbf{P}_{0:M}} \sum_{m=0}^{M} \sum_{k=0}^{K} l(\mathbf{x}(t_k; \mathbf{P}_{\mathrm{m}}), \mathbf{u}(t_k; \mathbf{P}_{\mathrm{m}}))$$
$$\text{s.t.} \begin{cases} \mathbf{x}_0 = \mathbf{x}(t_0; \mathbf{P}_0) & \\ \mathbf{x}_T = \mathbf{x}(t_K; \mathbf{P}_M) & \\ \mathbf{x}_{\min} \leq \mathbf{x}(t_k; \mathbf{P}_m) \leq \mathbf{x}_{\max}, & \forall m \in \mathcal{M}, k \in \mathcal{K} \\ \mathbf{u}_{\min} \leq \mathbf{u}(t_k; \mathbf{P}_m) \leq \mathbf{u}_{\max}, & \forall m \in \mathcal{M}, k \in \mathcal{K} \\ 0 \leq \mathbf{o}(\mathbf{x}(t_k; \mathbf{P}_m)), & \forall m \in \mathcal{M}, k \in \mathcal{K} \\ \mathrm{A}\mathbf{x}(t_K; \mathbf{P}_m) = \mathrm{A}\mathbf{x}(t_0; \mathbf{P}_{m+1}), & \forall m \in \mathcal{M} \setminus \{M\} \\ 0 = \delta(\mathbf{P}_m, \mathbf{P}_{m+1}), & \forall m \in \mathcal{M} \setminus \{M\} \end{cases} \tag{10}$$

where $\mathcal{M} = \{0, \ldots, M\}$ and $\mathcal{K} = \{0, \ldots, K\}$. The formulation further contains the constraint on the initial and final state, x_0 and x_T, the path inequality constraints with explicit obstacle constraints and finally two constraints related to the switch equality constraints, $\mathbf{h}_{\text{switch}}$.

The geometric switch constraints and obstacle constraints are detailed next.

1) Geometric switch constraint: First, we fix the end positions of each spline $\boldsymbol{\xi}_m(1)$, to the first position of the next spline $\boldsymbol{\xi}_{m+1}(0)$. This is established by equating a subset from state, $\mathbf{x}$, using matrix A selecting the first two elements.

Second we need to ensure that the orientation of the slider at the end of the trajectory matches, or that it is at an angle of 90°. Instead of computing the orientation of the slider, ϕ, at the start and end of each spline, we have that the orientation is defined by the orientation of the velocity vector.

Then we rely on an interesting property of clamped bivariate B-splines [9]. It can be shown that the derivative at the end nodes is then tangent to the vector between the final control points. The incoming velocity is proportional to

$$\vec{\mathbf{p}}_{m,n} = \mathbf{p}_{m,n} - \mathbf{p}_{m,n-1} \tag{11}$$

with $\mathbf{p}_{m,n} \in R^2$ reffering to the nth element of the mth spline. Whereas the outgoing velocity vector is proportional to

$$\vec{\mathbf{p}}_{m+1,0} = \mathbf{p}_{m+1,1} - \mathbf{p}_{m+1,0} \tag{12}$$

With this insight, the necessary constraint can be written as the product of the scalar product and cross product of both vectors.

$$\delta(\mathbf{p}_m, \mathbf{p}_{m+1}) = (\vec{\mathbf{p}}_{m,n} \cdot \vec{\mathbf{p}}_{m+1,0}) \cdot (\vec{\mathbf{p}}_{m,n} \times \vec{\mathbf{p}}_{m+1,0}) = 0 \tag{13}$$

The resulting function is continuously smooth, returning zero if either the scalar or cross product equals zero. This occurs whenever the vectors are orthogonal or aligned in parallel respectively. Further remark that this constraint does not enforce a certain switch.

2) Obstacle avoidance: The pusher-slider needs to be able to reach a goal position without colliding with any of the obstacles in the environment. In this work we model each obstacle as a circle or ellipsoid so that the distance between the pusher-slider and an arbitrary obstacle, $o_l \in \mathcal{E}$, with $l \in \{1, \ldots, n_o\}$ can now be quantified using a continuous metric, $d(\mathbf{x}, o) : \mathbb{R}^2 \times \mathcal{E} \to \mathbb{R}$, quantifying the Euclidean distance between the circumference of the slider and the obstacle.

In the case of an ellipsoidal obstacle the distance metric is defined as follows

$$d(\mathbf{x}, o) = \frac{(\Delta x)^2}{w^2} + \frac{(\Delta y)^2}{h^2} - 1 \tag{14}$$

where the distances $(\Delta x, \Delta y)$ are calculated as

$$(\Delta x, \Delta y) = \mathrm{R}(\theta) \cdot (x - x_e, y - y_e) \tag{15}$$

with $R(\theta)$ the planar rotation matrix. Here x_e and y_e represent the obstacles origin, θ its planar orientation and w and h denoting half the ellipse's width and height. Note that the ellipse's width and height account for both the geometry of the obstacle and the slider.

The vector, $\mathbf{o}$, is then defined as

$$\mathbf{o}(\mathbf{x}) = \begin{pmatrix} d(\mathbf{x}, o_1) & \ldots & d(\mathbf{x}, o_{n_o}) \end{pmatrix}^\top \tag{16}$$

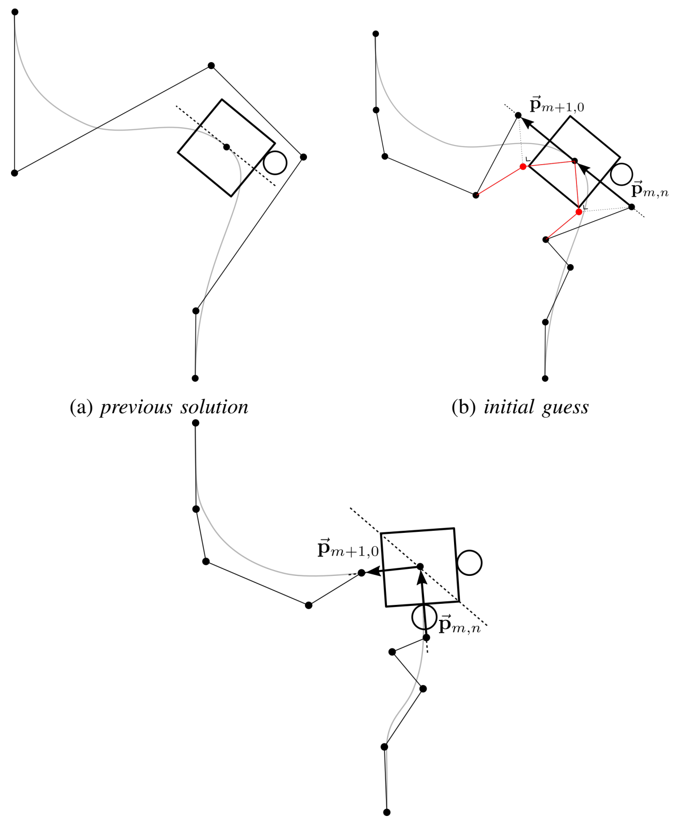
E. NLP initialization

As described in algorithm 1, the nonlinear program (10) solver is started with an initial guess for the control points $\mathbf{P}_{0:M}^{\text{init}}$. Two initialization methods are described, linear interpolation and an inverse calculation procedure.

1) Linear interpolation: Here the control points $\mathbf{P}_{0:M}$ are interpolated linearly between the initial state $\mathbf{x}_0$ and goal state, $\mathbf{x}_T$. This is a rather simple way to start the solver, with the benefit that no prior information is needed. Note that when this method is used with $M > 0$, this implies that the problem is reinitialized entirely, with the exception that the number of B-splines used to represent the smooth motion is now increased.

2) Inverse calculation: The second method of initialization is more involved and relies on the solution from a previous iteration. If in a previous solution a high geometric curvature is detected, then a switch in pushing direction can be needed. In this case, the control points are initialized in such a way that a switch in the vicinity of the high curvature section is likely. While the optimizer may autonomously identify and implement a switch without specific initialization, this approach often leads to extended solving times and can lack reliability. For that reason, a guided initial guess is beneficial for the overall performance of the algorithm.

A switch is expected to occur at the location of highest curvature along the spline. To allow for such a maneuver, two splines should connect at the switch point so that the orientation of the slider before and after the switch can be orthogonal. We create a proper initial guess in two steps. First, two splines are created to replace the previous solution. We then compute the new control points so that the two connecting splines approximate the original spline. Subsequently, the end and start control point of the spline before and after the switch are projected so that the control point polygon is orthogonal at the switch point. This process is illustrated in Fig. 2.



(a) *previous solution* (b) *initial guess*

(c) *projected guess*

Fig. 2: *Re-initializing the optimization with an additional spline to initialize the next optimization routine. The previous solution 2a is reused to estimate the spline parameters of the newly added splines 2b. Based on the curvature location and direction, the control points entering and leaving the switch point are projected so that the resulting angle of attack between both vectors is at* 90° *2c.*

With the previous solution $\boldsymbol{\xi}_m^{\text{prev}}$, the new connected splines, denoted $\boldsymbol{\xi}_m^{\text{new}}$ and $\boldsymbol{\xi}_{m+1}^{\text{new}}$, are initialized to approximate the $\boldsymbol{\xi}_m^{\text{prev}}$ trajectory. From the B-spline definition (4) it is clear that the B-spline is linear with respect to the control points. This means that we can express a vector of B-spline evaluations as a linear system in function of the control points.

$$\left(\boldsymbol{\xi}(\tau_0)^\top \quad \dots \quad \boldsymbol{\xi}(\tau_n)^\top\right)^\top = \mathrm{N} \otimes \mathrm{I}_2 \cdot \mathbf{P} \tag{17}$$

Here N constitutes the matrix of basis functions evaluated on the sampled points, $\{\tau_0, \dots, \tau_n\}$ with entries $\mathrm{N}_{ij} = b_{j,d}(\tau_i)$. When we choose n distinct sample points, the equation can be solved for $\mathbf{P}$. Remark that the sample points need not coincide with the knot vector. A simple approach is then to evaluate $\boldsymbol{\xi}_m^{\text{prev}}$ on an equidistant grid and fitting the spline control points of the new splines to match the previous output. With the switch point detected at τ^{switch}, we need to evaluate the spline $\boldsymbol{\xi}_m^{\text{prev}}(\tau)$ on an equidistant grid on $[0, \tau^{\text{switch}}]$ to compute the control points of $\boldsymbol{\xi}_m^{\text{new}}$ and on $[\tau^{\text{switch}}, 1]$ for $\boldsymbol{\xi}_{m+1}^{\text{new}}$.

For most cases this works well. However, in some cases, especially those trajectories which are relatively long and contain a large amount of curvature, this method can result in inaccurate derivatives of the spline at the endpoints. For this reason we slightly modify the previously discussed technique. We propose to drop one of the points in the grid and also include a constraint on the spline tangent at the switch point. The derivative of a clamped B-spline at the end points is tangent to the difference in the control points. An analogous set of control points for the B-spline, $\boldsymbol{\xi}_{m+1}$, can be determined by dropping one of the rows in (17) and adding the derivative into the set of equations

$$\begin{aligned} \left(\boldsymbol{\xi}_{m+1}(\tau_0)^\top \quad \dots \quad \boldsymbol{\xi}_{m+1}(\tau_{n-1})^\top\right)^\top &= \mathrm{N} \otimes \mathrm{I}_2 \cdot \mathbf{P}_{m+1} \\ \boldsymbol{\xi}'_{m+1}(\tau_0) &= \mathbf{p}_{m+1,1} - \mathbf{p}_{m+1,0} \end{aligned} \tag{18}$$

Here the sequence $\{\tau_0, \dots, \tau_{n-1}\}$ contains $n-1$ points in the range $[\tau_{\text{switch}}, 1]$. Together with the second equation we again have n equations which are all linear in the control points. Hence the control points are uniquely determined. An analogous case can be made for the spline $\boldsymbol{\xi}_m$ on $[0, \tau_{\text{switch}}]$.

The previously computed control points are still tangent at the switch point. To incite a switch by the optimizer, the second $\mathbf{p}_{m+1,1}$ and the second to last control point $\mathbf{p}_{m,n-1}$, of $\boldsymbol{\xi}_{m+1}$ and $\boldsymbol{\xi}_m$ respectively, are projected to be at 90°. This is also indicated in Fig. 2b with the resulting projection in Fig. 2c. The projected control point is computed by projecting $\vec{\mathbf{p}}$ defined as (11) and (12), for $\boldsymbol{\xi}_m$ and $\boldsymbol{\xi}_{m+1}$ respectively:

$$\vec{\mathbf{p}}_{\text{proj}} = \frac{\vec{\mathbf{p}}^\top \mathbf{v}}{\vec{\mathbf{p}}^\top \vec{\mathbf{p}}} \vec{\mathbf{p}} \tag{19}$$

The vector, $\mathbf{v}$, is the desired direction of each spline at the switch point. The vector is computed by rotating the tangent of the spline at the switch point, $\boldsymbol{\xi}'(\tau^{\text{switch}})$, 45° and 135° for $\boldsymbol{\xi}_{m+1}$ and $\boldsymbol{\xi}_m$ respectively.

F. Geometric optimization

In this work we focus on the geometry of the problem while keeping the time parameterization of the trajectory fixed. Due to the specific flat properties of the pusher-slider system, the optimization problem can be decomposed into a geometric and time optimization problem [8]. Hence, once a geometric solution is determined, a velocity profile can be imposed subsequently. In the experiments in this work the time horizon is chosen as T=1.

IV. Simulation Experiments

To test the effectiveness of our methodolgy, several environments with a number of obstacles are created. We start by outlining the experimental details and continue by discussing the results while highlighting the different behaviours we encountered.

A. Experimental details

Within each case, the objective remains consistent: to transition from the initial state $\mathbf{x}_0 = (0,0,0,0)^\top$ to the desired goal state $\mathbf{x}_T = (x, y, \phi, c)$, all while ensuring that no collisions occur with the obstacles. The objective is encoded through the problem formulation (10) with cost objective

$$l(\mathbf{x}, \mathbf{u}) = \mathbf{x}^\top \mathrm{Q} \mathbf{x} + \mathbf{u}^\top \mathrm{R} \mathbf{u} \tag{20}$$

We add an additional regularization term in function of the control points which proved beneficial for the convergence of the optimizer. The inclusion of this term serves the purpose of ensuring a uniform distance is maintained between the control points.

$$l_{\text{reg}} = w_{\text{reg}} \sum_{m=0}^{M} \sum_{i=0}^{n} \|\mathbf{p}_{m,i+1} - \mathbf{p}_{m,i}\|^2 \tag{21}$$

The following numerical values were chosen: $Q = \text{diag}(10^{-4}, 10^{-4}, 0, 0)$, $R = \text{diag}(1, 0.01), K = 20, w_{\text{reg}} = 10$ and $n = 5$. The optimization problem is constructed with CasADi [27] and solved using IPOPT [28]. All computations are run on a 64-bit AMD Ryzen 5 2600x workstation with 32GB RAM.

B. Results

We generate several scenarios with a set of densely arranged obstacles on which the algorithm 1 is deployed. For each set-up the same initialization parameters are used. The algorithm is started with a single spline, i.e. $M = 0$, where the control points are linearly interpolated between start $\mathbf{x}_0$ and goal state $\mathbf{x}_T = (x, y, \phi, c)$. The resulting trajectories are shown in Fig. 3 and Fig. 4. Depending on the environment, the algorithm converges to a certain number of splines $M+1$. If few obstacles are present, the solver will find a solution with just one spline and no switching behaviour. In the environment shown in Fig. 3a one spline does not suffice. Initially the algorithm succeeds in finding a feasible solution with $M = 0$. However due to the densely arranged obstacles, the resulting solution exceeds the allowable curvature. Subsequently, the number of splines is increased by one, and the solver is reinitialized with orthogonal control points near the high curvature section. Finally, the solver converges to a solution with two splines which implements a switch to traverse the narrow passage.

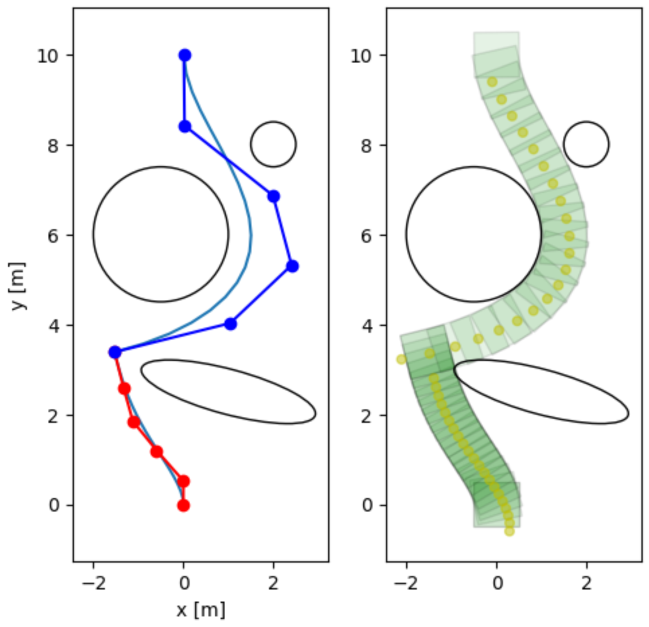
In Fig. 3b, the algorithm is again started with a single spline. In this case however the solver is unable to find a feasible solution. In the next iteration the number of splines is increased to increase the expressiveness. Similar to the previous case, a high geometric curvature is encountered and the solver is restarted with an additional spline and proceeds to implement a switch. In the subsequent iteration, the solver again encounters a high geometric curvature. Since the final two splines are connected without utilizing a switch, the solver is reinitialized before converging to the final solution which implements two switches.

Since a fixed hyperparameterization of each B-spline was used, i.e. fixed number of knots and degree, some cases require an increased number of splines to increase expressiveness without adding additional switches. In Fig. 4 the algorithm converges to a solution with three splines which contains an unused switch in between the two final splines.

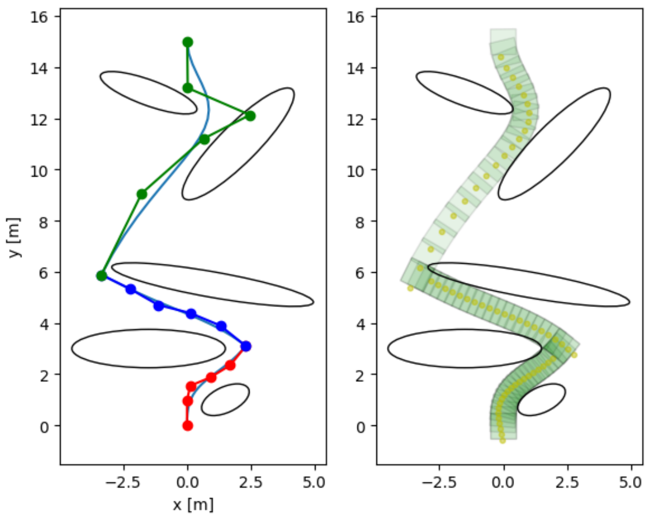
Whenever the number of splines are increased, so does the number of optimization variables and the computational requirement of the problem. In most cases the algorithm converges relatively fast. The computational requirement is reported in Fig. 3 and Fig. 4.

V. Conclusion

In this work we tackled the challenge of controlling a sliding object with non-prehensile manipulation through a cluttered environment. To solve the associated TAMP problem, we proposed an algorithm to automatically determine both a feasible pushing sequence and trajectory through a cluttered environment. Given a specific pushing sequence, the TAMP problem is transcribed, with a set of interconnected B-splines, into a trajectory optimization problem. The resulting trajectory optimization problem jointly optimizes the pusher-slider trajectory and switching parameters such as switching location and orientation. By detecting high curvature, the pushing sequence is adapted and the associated trajectory optimization problem is resolved. This process is repeated iteratively before converging to a solution. The algorithm was deployed in a number of simulation environments, demonstrating it's ability to efficiently and automatically find feasible trajectories through cluttered environments by incorporating switches in pushing direction mid trajectory.



(a)



(b)

Fig. 3: *Trajectories in different scenarios with goal state* $\mathbf{x}_T = (0, 10, 0, 0)$ *for Fig. 3a and* $\mathbf{x}_T = (0, 15, 0, 0)$ *for Fig. 3b.The solution of 3a and 3b took the algorithm 470ms and 4.30s to converge respectively.*

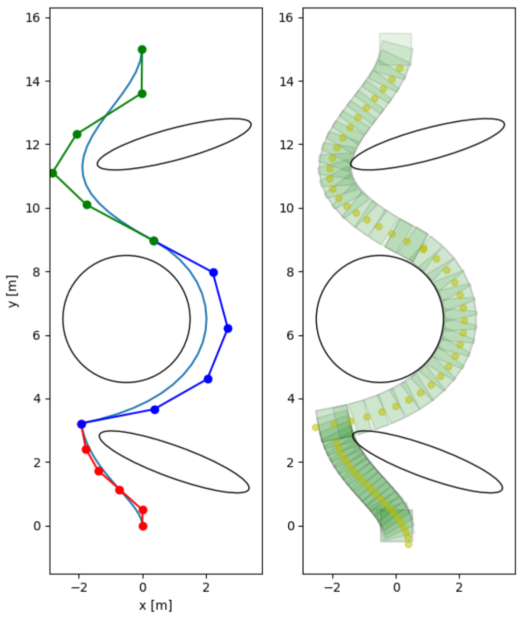


Fig. 4: *Trajectory solution with goal state* $\mathbf{x}_T = (0, 15, 0, 0)$. *The algorithm converges to multiple splines where one possible switch is left open and connected as a continuous trajectory. The algorithm took 2.29s to converge.*

In future work we would like to extend the method to longer horizon problems, with the inclusion of a larger set of distinct actions. Validating the method on an experimental setup by incorporating it in an MPC framework would be another track.

Acknowledgment

This work was supported by the Flanders Make projects DIRAC and the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

References

[1] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. IEEE Transactions on Robotics, 36(4):1307–1319, 2020.

[2] Fabio Ruggiero, Vincenzo Lippiello, and Bruno Siciliano. Nonprehensile dynamic manipulation: A survey. IEEE Robotics and Automation Letters, 3(3):1711–1718, 2018.

[3] J Zachary Woodruff and Kevin M Lynch. Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 4066–4073. IEEE, 2017.

[4] Francois Robert Hogan and Alberto Rodriguez. Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. CoRR, abs/1611.08268, 2016.

[5] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In 2016 IEEE/RSJ international conference on intelligent robots and systems (IROS), pages 30–37. IEEE, 2016.

[6] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In Robotics: Science and Systems XIV. Robotics: Science and Systems Foundation, June 2018.

[7] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In 2014 IEEE-RAS International Conference on Humanoid Robots, pages 279–286, 2014.

[8] Tom Lefebvre, Sander De Witte, Thomas Neve, and Guillaume Crevecoeur. Differential Flatness of Slider–Pusher Systems for Constrained Time Optimal Collision Free Path Planning. Journal of Dynamic Systems, Measurement, and Control, 145(6), 04 2023. 061001.

[9] Florin Stoican, Ionela Prodan, and Dan Popescu. Flat trajectory generation for way-points relaxations and obstacle avoidance. In 2015 23rd Mediterranean Conference on Control and Automation (MED), pages 695–700. IEEE, 2015.

[10] Nicolas Petit, Mark B. Milam, and Richard M. Murray. Inversion based constrained trajectory optimization. IFAC Proceedings Volumes, 34(6):1211–1216, July 2001.

[11] Richard M. Murray, John Hauser, Ali Jadbabaie, Mark B. Milam, Nicolas Petit, William B. Dunbar, and Ryan Franz. Online Control Customization via Optimization-Based Control, page 149–174. Wiley, 1 edition, April 2003.

[12] François Chaplais and Nicolas Petit. Inversion in indirect optimal control of multivariable systems. ESAIM: Control, Optimisation and Calculus of Variations, 14(22):294–317, April 2008.

[13] Ngoc Thinh Nguyen, Ionela Prodan, Florin Stoican, and Laurent Lefèvre. Reliable nonlinear control for quadcopter trajectory tracking through differential flatness. IFAC-PapersOnLine, 50(1):6971–6976, 2017. 20th IFAC World Congress.

[14] Melissa Greeff and Angela P. Schoellig. Flatness-based model predictive control for quadrotor trajectory tracking. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 6740–6745, 2018.

[15] Zejiang Wang, Jingqiang Zha, and Junmin Wang. Flatness-based model predictive control for autonomous vehicle trajectory tracking. In 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pages 4146–4151, 2019.

[16] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Kinodynamic trajectory optimization and control for car-like robots. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5681–5686, 2017.

[17] Neel Doshi, Francois R. Hogan, and Alberto Rodriguez. Hybrid differential dynamic programming for planar manipulation primitives. In 2020 IEEE International Conference on Robotics and Automation (ICRA), page 6759–6765, May 2020.

[18] João Moura, Theodoros Stouraitis, and Sethu Vijayakumar. Non-prehensile planar manipulation via trajectory optimization with complementarity constraints. In 2022 International Conference on Robotics and Automation (ICRA), page 970–976, May 2022.

[19] Yongpeng Jiang, Yongyi Jia, and Xiang Li. Contact-aware non-prehensile manipulation for object retrieval in cluttered environments. September 2023.

[20] Jiaji Zhou, Yifan Hou, and Matthew T Mason. Pushing revisited: Differential flatness, trajectory planning, and stabilization. The International Journal of Robotics Research, 38(12–13):1477–1489, October 2019.

[21] Thomas Neve, Tom Lefebvre, Sander De Witte, and Guillaume Crevecoeur. Flatness-based mpc using b-splines transcription with application to a pusher-slider system. In 2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), pages 132–137, 2023.

[22] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning.

[23] Contributions to the problem of approximation of equidistant data by analytic functions. Quarterly of Applied Mathematics, 4:112–141, 1946.

[24] Boris Rohal'-Ilkiv, Martin Gulan, and Peter Minarčík. Implementation of continuous-time mpc using b-spline functions. In 2019 22nd International Conference on Process Control (PC19), pages 222–227, 2019.

[25] Carl de Boor. On calculating with b-splines. Journal of Approximation Theory, 6(1):50–62, 1972.

[26] Simon Helling, Max Lutz, and Thomas Meurer. Flatness-based mpc for underactuated surface vessels in confined areas. IFAC-PapersOnLine, 53(2):14686–14691, 2020. 21st IFAC World Congress.

[27] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. Mathematical Programming Computation, 11(1):1–36, 2019.

[28] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program., 106(1):25–57, mar 2006.