PNC-CL a toolbox for piecewise affine convex-liftings with application in planning and control design

Sorin Olaru, Turan Konyalıoğlu, Marius Jaegler, Mickael Martinelli, Valentin Carrez

PNC-CL a toolbox for piecewise affine convex-liftings with application in planning and control design

Sorin Olaru[1], Turan Konyalioglu[1], Marius Jaegler, Mickaël Martinelli and Valentin Carrez

***Abstract*— The paper describes a novel, publicly available toolbox for piecewise-affine (PWA) convex-lifting. The main goal is to provide a tool for efficient partitions of sets in finite-dimensional spaces with applications in navigation and control design. This objective is achieved by employing a lift-and-project philosophy that relies on performant linear programming optimization and polyhedra manipulation. Once a PWA lifting is constructed, the related polyhedral partition provides an interconnection graph that can be used for path planning, which is considered here as the main application. The paper provides the main theoretical elements of the technique's foundation and describes the toolbox's structure along with the numerical artifacts used for its implementation. Illustrative examples for the practical problems are provided, supporting the claims of efficiency and versatility.**

I. INTRODUCTION

Lift and project methods are used in several design techniques in order to avoid the inherent complexity of the original problem formulation. In a nutshell, the idea is to transform the problem into a higher space of decision variables. This increase in the dimension (lifting) is done in such a way to improve the structure of the problem to be solved, either in terms of convexity, symmetry or interdependence. To retrieve the solution of the original problem, a projection operator is used to make available the solution and its properties in the original decision space.

Lift-and-project can be traced back as an effective technique in the field of combinatorial optimization in the late 1970s and 1980s [1], [2], [3]. It differs from traditional optimization in that it does not represent the combinatorial decision problem directly in the original integer variables. Instead, it restructures the problem in a higher-dimensional space (lift operation) with relaxed, continuous variables. This transformation effectively converts the initial integer program into a linear program with clear advantages for effectiveness. In a different field of applications (e.g. robotics), the Koopman operators are used as an elevation of the dynamics of the nonlinear system into a higher-dimensional space where linearity can be exploited. Reproducing Kernel Hilbert Spaces (RKHS) represents another lift-and-project method mainly employed in the data-driven approach for the analysis or the control design for nonlinear dynamical systems. Upon lifting the relevant nonlinear system to the RKHS feature space, it assumes a linear character and operations can be performed efficiently, followed by projection back to the original nonlinear system [4], [5].

[1] Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des Signaux et Systèmes, France. Email: sorin.olaru@centralesupelec.fr turan.konyalioglu@centralesupelec.fr

In control design, the polyhedral partitions of finite-dimensional spaces are a commonly encountered problem emerging either from obstacles description within the operation space or from the piecewise description of the dynamics or of the (constrained) control. The lift-and-project approach in this framework intends to exploit the piecewise affine (convex) functions as an effective way to represent polyhedral partitions. Piecewise affine convex lifting has been shown to provide an efficient implementation approach for the piecewise affine (PWA) controllers, and its utility has expanded beyond the realm of PWA controllers in recent developments dealing with fragility or inverse optimality topics [6], [7].

In path planning, the polyhedral partitions can be used to represent the operation space containing obstacles. In the case of a cluttered environment, the density of the obstacles leads to a complex partition and the polyhedral operation are often complex due to the interdependence of the regions' geometry and neighborhood properties. To avoid this complexity, a lifting in a higher dimensional space using PWA functions allows us to deal with the interconnections in an efficient manner by exploiting the epigraph properties.

The theoretical properties of the PWA convex liftings have been explored in the literature [8] and subsequently used for path planning [9], but to this day, there is no effective, publicly available toolbox able to deal with the convex lifting problems. The present paper aims to address this gap and describes a collection of routines that are able to solve basic PWA convex-lifting problems and in particular, can be used for path planning in cluttered environments with particularly efficient execution time. From the implementation point of view, several contributions are described with respect to the numerical methods implemented. Also, an effort is made here to describe the configuration of the toolbox in terms of necessary input data, auxiliary computation routines (optimization and polyhedral manipulation routines) and the structure of the data flow.

Next we will provide a comprehensive overview of the definitions, construction conditions, existing conditions, and key concepts related to PWA convex lifting in Section II. Section III is dedicated to the toolbox organization and practical guidelines for its effective use. The contributions from the methodological and numerical point of view with respect to the state of the art are described in Section IV. Illustrative examples of the usefulness of the technique are provided in Section IV before drawing the conclusions.

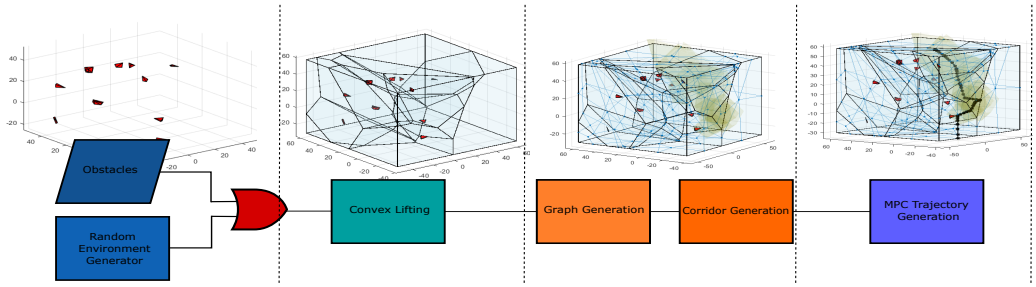


Fig. 1: CL-PWA toolbox module structure.

II. Theoretical Background and Objectives

Building convex liftings and using them in applications significantly depends on polyhedral computations. A series of formal definitions in this respect are introduced next.

Definition 1: The intersection of a finite number ($m \in \mathbb{Z}^+$) of halfspaces defines a polyhedron:

$$P = \left\{x \in \mathbb{R}^d | Hx \leq w\right\}$$

with $H \in \mathbb{R}^{m\times d}$ describing the orientation of constraints and $w \in \mathbb{R}^m$ the vector of offsets for each of them.

Definition 2: Given a bounded polyhedral set $X \subset \mathbb{R}^d$ defining a global restriction, the family of polyhedral subsets of X is defined as:

$$\mathbb{P}(X) = \{P \subseteq X | \exists m \in \mathbb{N}, H \in \mathbb{R}^{m\times d}, w \in \mathbb{R}^{m_i}, \text{ such that } x \in P \Leftrightarrow Hx \leq w\}.$$

A cluttered environment refers to the space in which a large amount of obstacles are located, which restricts the motion of an agent or a robot. It is possible to define those obstacles as a collection of disjoint polyhedra. Thus, any disjoint combination of polyhedra, spatially distributed in a finite-dimensional compact space, X, is characterized below.

Definition 3: In a cluttered environment X, all possible collections of N disjoint polyhedral sets is denoted as:

$$\mathbb{P}_D^N(X) = \{\{P_i\}_{i\in\mathcal{I}_N} \subset \mathbb{P}^N(X) \text{ such that } \operatorname{int}(P_i) \cap \operatorname{int}(P_j) = \emptyset,\ (i,j) \in \mathcal{I}_N^2\}.$$

From a distribution of disjoint obstacles, one can induce a partition of X.

Definition 4: Given a collection of obstacles, $\mathcal{P} \triangleq \{P_i\}_{\mathcal{I}_N} \subset \mathbb{P}_D^N(X)$, within a cluttered environment X, a family of space partitions generated by these objects can be defined as:

$$\mathbb{W}^N\left(\{P_i\}_{\mathcal{I}_N}, X\right) = \{\{X_i\}_{\mathcal{I}_N} \subset \mathbb{P}_D^N(X) \,|X = \bigcup_{i\in\mathcal{I}_N} X_i, P_i \subseteq X_i, \forall i \in \mathcal{I}_N\}. \quad (1)$$

Once these definitions are introduced, the mathematical objective of the toolbox developped can be stated readily: given a set of obstacles $\{P_i\}_{\mathcal{I}_N}$ cluttering the environement X, construct a partition in $\mathbb{W}^N\left(\{P_i\}_{\mathcal{I}_N}, X\right)$.

The effective tool in this endeavour is the *convex lifting*. The lifting operation over a collection of disjoint obstacles refers to the construction of an affine function whose epigraph represents a polyhedral region in a higher dimensional space. The faces of this polyhedron are in one-to-one relationship with the original obstacles.

Definition 5: For the given collection of polyhedral obstacle sets $\mathcal{P}$, a piecewise affine function $z : X \to \mathbb{R}$, explicitly represented as

$$z(x) = \max_{i\in\mathcal{I}_N} a_i^T x + b_i \quad (2)$$

designates a *convex lifting function* if it satisfies the property:

$$z(x) > a_j^T x + b_j,\ \forall x \in \operatorname{int}(P_i),\ (i,j) \in \mathcal{I}_N^2.$$

The natural question is whether such a lifting can be created effectively. The theoretical foundations addressing this question have been studied in [8]. In a nutshell, solving the following optimization problem in (3) one can obtain the coefficients of the PWA function (2).

$$\min_{a_i,b_i} J = \sum_{i=1}^{N} \|a_i^T \quad b_i\|_2^2 \quad (3a)$$

$$\text{s.t. } a_i^T v + b_i \leq M, \forall v \in \mathcal{V}(P_i), \forall i \in \mathcal{I}_N, \quad (3b)$$

$$a_i^T v + b_i \geq a_j^T v + b_j + \epsilon, \forall v \in \mathcal{V}(P_i), (i,j) \in \mathcal{I}_N^2 \quad (3c)$$

where $M, \epsilon > 0$ are boundedness and convexity parameters. The optimization problem is subjected to boundedness constraints as in (3b), which revokes inclusion property and convexity constraints (3c). The epigraph of the computed lifting function represents a polyhedral set,

$$\mathcal{L} = \left\{ \begin{bmatrix} x \\ z \end{bmatrix} \in \mathbb{R}^{d+1} : \begin{bmatrix} a_i^T & -1 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \leq -b_i, i \in \mathcal{I} \right\} \quad (4)$$

Recalling the ultimate objective is the construction of a space partition $\mathcal{X} \triangleq \{X_i\}_{\mathcal{I}_N} \subset \mathbb{W}^N(\mathcal{P}, X)$, the effective implementation passes by the construction of polyhedral sets X_i obtained by projecting the facets of the lifted polyhedron $\mathcal{L}$ back into the original space, $\mathcal{X} \subset \mathbb{R}^d$.

$$X_i = \operatorname{proj}(\mathcal{F}_i(\mathcal{L}), \mathcal{X}), i \in \mathcal{I}_N \quad (5)$$

In Fig. 2, the steps of the lift-projection technique are depicted schematically. The yellow polyhedral sets (obstacles) are originally located in the planar space, while the purple partitions illustrate the epigraph and the resulting space partition.

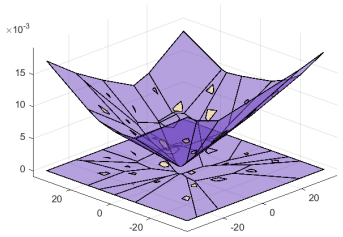


Fig. 2: Lifting and projection.

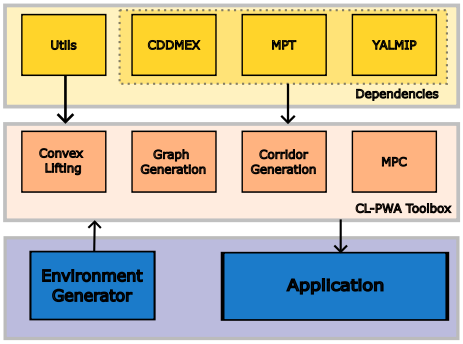


Fig. 3: Toolbox layers.

III. Structure of the Toolbox and Common Problems It Solves

This section is dedicated to explaining the utilization of the toolbox by pointing to the theoretical foundation of the main methods. The tool provides a safe navigation technique for obstacle avoidance in a cluttered environment. In Fig. 1, the tasks of the modules are illustrated in a 3D environment (although the methods are generalizable in any finite-dimensional space d). The toolbox has four main modules, and the trajectory is produced as solution evolves sequentially. Fig 3 presents the layers of the toolbox categorized in modules. The toolbox builds on dependencies with respect to CDDMEX [10], MPT [11], and YALMIP [12] for the operations such as polyhedron computations (vertex enumeration, minimum halfspace calculation, and other set operations), optimization interface. It also has a *utils* module containing some necessary common functions providing functionalities to the main toolbox modules. The aim is to adhere the toolbox structure to the architecture described in the figure.

A. The convex lifting implementation

The relevant framework produces a space partition by considering boundedness, convexity parameters, and obstacles in the form of polyhedrons. The toolbox accepts a list of obstacles that fulfill the Definition 3 as input. It has the capability to generate random obstacles within an environment with predefined space bounds. The user can generate a random cluttered environment using the following intuitive instructions with respect to the parameters:

```
dim = 3; nObs = 10; meanSize = 3; stdSize = 0;
stdLoc = 0; len = 50; maxIter = 50;
src = [-len/2 -len/2 -len/2];
P = randEnv(dim, nObs, meanSize, stdSize, stdLoc,
    len, maxIter)
```

The parameters, such as the number of obstacles, the space dimension, the average size of the obstacles, the standard deviation of obstacles' size, and their locations, can be adjusted in such a randomly generated scenario. Using the generated list of obstacles, "P" satisfying definition 3 that belong to the "Polyhedron" class provided by MPT toolbox [11] space partitioning can be generated as follows:

```
opt     = LiftOptions.convexDefault();
lifting = Lifting.find(P, opt);
X       = lifting.getPartition();
```

The "find" method solves the problem in (3) and returns an object that includes piecewise affine function parameters, space-partition, and a dialog indicating whether the optimization problem is solved successfully. Then, the "getPartition()" method computes the space partition.

B. Construction of the graphs

To address the global navigation problem in a cluttered environment, an interconnected graph can be constructed using the facets of the partitions produced by convex lifting.

Definition 6: The interconnected graph of paths is defined as $\mathcal{G}(\mathcal{N}, \mathcal{E}, f)$ where $\mathcal{N}$ represents the set of nodes corresponding to the vertices of the graph, $\mathcal{E}$ represents the set of edges, and $f : \mathcal{E} \to \mathbb{R}$ is a function that denotes the weights associated with each edge of the graph.

In the present toolbox, the default option for graph generation use facets of polyhedra as edges, $\mathcal{E}$, and their vertices as nodes, $\mathcal{N}$. In the following code, the first "EdgeGraphBuilder" class is called from the "graph" namespace to create an instance "gbuilder". Then, the "getGraph" method can be exploited on the lifting to construct the graph by taking gBuilder, and the working space "bbx" as inputs.

```
gBuilder = graph.EdgeGraphBuilder();
G = lifting.getGraph(gBuilder, bbx);
```

Each edge of the generated graph is a possible path for navigation with regard to the related weight function. Let us define the path segment and path radius.

Definition 7: Given a collection of polyhedral sets $\mathcal{P}$ and the related partition generated via convex-lifting, $\mathcal{X}$, a corridor between two nodes $(x_0, \bar{x}) \in \text{int}(\mathcal{X} \backslash \mathcal{P})$ is characterized by the existence of two functions: $\gamma : [0,1] \to (\mathcal{X} \backslash \mathcal{P})$ and $\rho : [0,1] \to \mathbb{R}_{>0}$ satisfying $\gamma(0) = x_0$ and $\gamma(1) = \bar{x}$, $\gamma(\theta) \oplus \mathbb{B}^2_{0,\rho(\theta)} \subset (\mathcal{X} \backslash \mathcal{P})$, $\forall \theta \in [0,1]$.
The radius is found by calculating the minimum distance between obstacles and the path segment.

$$\rho_i(\theta) = \min_{P_j \in \mathcal{P}} d(P_j, \gamma_i), \forall \tilde{\theta} \in [0,1] \tag{6}$$

```
G = corridors.corridor_width(G, obstacles);
G = corridors.edge_weight(G);
path = graph.path(G, src, dest, obstacles, X);
```

With the existence of a graph $\mathcal{G}(\mathcal{N}, \mathcal{E}, f)$, the shortest path can be found between two nodes, x_0 and x_f by using a

graph search technique such as the Dijkstra algorithm as follows: $\mathcal{Q}^p(x_0, x_f) = \mathcal{D}_G(x_0, x_f)$ where the resulting path, $\mathcal{Q}^p \in \mathbb{R}^{n\times p}$ is combination of nodes and p is the number of nodes. Furthermore, the length of the path can be denoted by $\delta(\mathcal{Q}^p(x_0, x_f) = \|x_0 - x_1\|_2 + \delta(\mathcal{Q}^{p-1}(x_1, x_f))$.

C. Safe corridor construction

While path generation addresses mainly the feasibility of the planing, corridors help manage navigation issues in cluttered environments, especially when considering dynamic constraints and uncertainties. Corridors act as hard constraints on the agent's trajectory, representing the collision-free space between obstacles and guiding navigation through the environment's configuration space by offering a feasible path and a (convex) constrained domain. Next, a series of corridor segments, safe of any obstacles in the configuration space, can be delineated as follows:

$$\Pi = \{\ x \in \mathbb{R}^d : \exists \theta \in [0,1] \,\text{s.t.}\ \ x \in \gamma(\theta) \oplus \mathbb{B}^2_{0,\rho(\theta)}\} \quad (7)$$

Computing corridors by considering the union of convex sets can simplify navigation tasks. Specifically, we define the corridor as the union of convex sets calculated for each segment of the piecewise linear path, $\Pi = \bigcup_{i=1}^{N_c} \Pi_i$ where,

$$\Pi_i = \{\ x \in \mathbb{R}^d : \exists \tilde{\theta} \in [0,1] \,\text{s.t.}\ \ x \in \gamma_i(\tilde{\theta}) \oplus \mathbb{B}_{0,\rho_i(\tilde{\theta})}\} \quad (8)$$

with $\gamma_i(0) = x_i$ and $\gamma_i(1) = x_{i+1}$. As demonstrated below, corridors are computed as below,

```
[P, min_width] = corridors.corridor_post_processing
    (G, path, n);
```

This function takes a graph and a path as inputs and returns a list of Polyhedron objects representing corridors and a minimum width parameter to ensure feasibility.

IV. Methodological contributions

In this section, we focus on the methodological contribution of this paper to the existing convex lifting framework.

A. Fast space-partitioning

The space-partitioning via (4) and (5) can be relatively costly in computation as it requires a sequence of polyhedral operations. The convexity constraints fulfilling the lifting function property lead to the an explicit halfspace formulation:

$$X_i = \{x \in \mathbb{R}^d | (a_j - a_i)^T x \leq (b_i - b_j), (i,j) \in \mathcal{I}_N\}. \quad (9)$$

Therefore, instead of computing the epigraph of the piecewise lifting function and then projecting it onto the navigation space, one can obtain the space partitioning in a faster way as in (9).

B. A novel convex-lifting m-formulation and implementation

In terms of construction and computation complexity and feasibility analysis, we pursue a better derivative of the optimization problem presented in (3). This approach involves transforming the quadratic cost function into a linear cost function and considering the convexity variable as an optimization parameter, as shown below:

$$\arg\min \quad -\epsilon \quad (10a)$$

$$\text{s.t. } |\epsilon| \leq C \quad (10b)$$

$$a_i^T v + b_i \geq a_j^T v + b_j + \epsilon, \forall v \in \mathcal{V}(P_i), (i,j) \in \mathcal{I}_N^2 \quad (10c)$$

where m is convexity parameters and $C \in \mathbb{R}^+$ is the bound for convexity parameter. A significant advantage of this approach, in terms of the feasibility of the optimization problem, is its ability to determine whether the problem is liftable or non-liftable. A negative ϵ solution indicates that the polyhedral set cannot be lifted convexly.

In the toolbox, the convex-lifting problem defined by (10) can be implemented by selecting the lifting settings accordingly,

```
opt     = LiftOptions.linearDefault();
lifting = Lifting.find(P, opt);
```

C. Shorter paths with hulling of cluttered region

The convex lifting framework aims for feasibility-guaranteed paths in an environment rather than computing optimal paths. Therefore, grouping a collection of regions in an environment that has obstacles cluttered can lead to partitions with a larger aspect ratio. In turn, as they avoid the group of obstacles as a whole, they would cause unnecessary maneuvers that extend the path distance. Even though the navigation strategy inside of a safe corridor allows agents to minimize the distance taken to a certain point, we also consider presenting a feature that shortens the path to be computed in the toolbox in the context of path planning. To have a measure of cluttered region, one can take the convex hull of polyhedron obstacles $\tilde{\mathcal{P}} = co(\mathcal{P})$ s.t. $\mathcal{P} \subset \mathbb{P}_D^N(\mathcal{X})$. Then, a margin can be computed by enlarging $\tilde{\mathcal{P}}$ with a homothetic factor as $\bar{\mathcal{X}} = \lambda\tilde{\mathcal{P}}$. The resulting polytope defines a new bounded space whereby each space partition can be recomputed as $\mathcal{X}_i = \mathcal{X}_i \cap \bar{\mathcal{X}}$. This feature can be activated by calling the function below:

```
bbx = util.bounding_polyhedron(P, convex_hull,
    margin);
```

where "P" is a collection of obstacles, "margin" is a homothetic factor, and "convex_hull" is a boolean for turning off this option.

D. Barycenteric graph generation

In the general case, the graph is generated by using edges of the polyhedral space partition and their vertices while ensuring connectivity between the initial node of an agent and the closest graph node to it. Another option for selecting the graph nodes can be the Barycenter of the polyhedral partitions. This functionality can be activated by calling the graph builder as follows:

```
gBuilder = graph.BarycenterGraphBuilder();
```

This choice comes with a more refined node population as it improves path distance. This method is formulated as:

$$\mathcal{N} = \bigcup_{k=1}^{n-1} \bigcup_{P_i \in \mathcal{F}^{k-1}(\mathcal{P})} \bigcup_{E \in \mathcal{F}^{k}(P)} (\mathcal{B}(P_i) \cup \mathcal{V}(E))$$

$$\mathcal{E} = \bigcup_{k=1}^{n-1} \bigcup_{P_i \in \mathcal{F}^{k-1}(\mathcal{P})} \bigcup_{E \in \mathcal{F}^{k}(P_i)} \{\mathcal{B}(P_i), \mathcal{B}(E)\}$$

where n is the dimension of the space, $\mathcal{F}$ refers to the facet operation, and superscript is the power of the facet operation.

V. MPC-BASED TRAJECTORY GENERATION

The main advantage of the safe corridors is that they substitute non-convex obstacle avoidance constraints by a piecewise convex sequence of constraints. Therefore, as long as the system dynamics remain linear and system constraints stay convex, we can exploit convex MPC techniques for navigation purposes. To commence, consider an MPC formulation with linear time-invariant (LTI) dynamics of an agent:

$$\begin{aligned}\mathcal{J}(N_p, \bar{x}, x_k, U) = \|x_{k+N_p|k} - \bar{x}_i\|_P^2 + \\ \sum_{l=1}^{N_p-1} \|x_{k+l|k} - \bar{x}_i\|_Q^2 + \sum_{l=1}^{N_p-1} \|\Delta u_{k+l|k}\|_R^2\end{aligned} \tag{11}$$

where N_p is prediction horizon, $\bar{x}_i$ is reference state in a corridor segment, Q is state cost matrix, R is control increment cost matrix, and P is the terminal cost matix. The vector $U = \begin{bmatrix} u_{k|k} & \dots & u_{k+N_p-1|k} \end{bmatrix}^{\mathrm{T}}$ is the optimization argument:

$$\mathcal{T}(\Pi_i, N_p, \mathcal{X}_f, \bar{x}_i, \mathcal{X}) : \min_U \mathcal{J}(N_p, \bar{x}_i, x_k, U) \tag{12a}$$

$$\text{s.t.} \quad x_{k+l+1|k} = Ax_{k+l|k} + Bu_{k+l|k}, \tag{12b}$$

$$u_{k+l|k} \in \mathcal{U}, \ \forall l = 1 : N_p - 1, \tag{12c}$$

$$x_{k+l|k} \in \tilde{\Pi}_i \tag{12d}$$

$$x_{k+N_p|k} \in \mathcal{X}_f(\bar{x}_i) \tag{12e}$$

with state-space dynamics (12b), input constraints (12c) and state constraints (12d). Also, corridor constraints in the state-space can be defined as,

$$\tilde{\Pi}_i = \{x \in \mathbb{R}^n \mid Cx \in \Pi_i\} \tag{13}$$

To ensure the system remains within the corridors, the final state of the trajectory is required to end in a controlled invariant set, as revoked by (12e), denoted as $\tilde{\mathcal{X}}$, such that $(A + BK)\tilde{\mathcal{X}} \subseteq \tilde{\mathcal{X}}$, assuming the (A, B) pair is controllable [13]. The feedback control gain K, which stabilizes the system, can be determined by solving the discrete Riccati equation. The state constraint set associated with the control admissible and output constraint set can be defined as in (14).

$$\tilde{\mathcal{X}}(C, \mathcal{Y}, K, \mathcal{U}) = \{x \in \mathbb{R}^n \mid Cx \in \mathcal{Y}, \text{ and } Kx \in \mathcal{U}\} \tag{14}$$

Then, with the existence of a stabilizing feedback gain, a controlled invariant set can be established as in (15) by computing the maximal admissible set [14].

$$\Omega(A_c, \tilde{\mathcal{X}})_\infty = \{x \in \mathbb{R}^n \mid A_c^k x \in \tilde{\mathcal{X}}, \ \forall k \in \mathbb{Z}^+\} \tag{15}$$

where $A_c = A + BK$ is the stabilized system closed-loop state matrix. During the corridor transition, an agent can guarantee to remain inside the corridor by reaching a controlled invariant set defined by Ω_i. In the toolbox, the controlled invariant set is calculated by using the "mpc" class' method, "controlInvariant()", which in turn calls the "maximumInvariantSet()" method to compute the maximum admissible set. In the meantime, the controlled invariant set calculation can be performant in relation to the complexity of the computed state constraint set and agent dynamics. Due to this, the toolbox also offers fast controlled invariant set computation by shifting and enlarging a pre-computed controlled invariant set, $\mathcal{X}_f(0) = \Omega(A_c, \mathcal{S})_\infty$ where $\mathcal{S} = \tilde{\mathcal{X}}(C, S, K, U)$ with $S \subset \tilde{\Pi}_1 \cap \tilde{\Pi}_2$. Then, it can be shifted to the position, $\bar{x}_i \in \tilde{\Pi}_i \cap \tilde{\Pi}_{i+1}$ and enlarged as solving the optimization problem in (16), consecutively for each corridor transition.

$$\max_\lambda \lambda \text{ s.t. } \lambda \mathcal{X}_f(0) \subset (\tilde{\Pi}_i \cap \tilde{\Pi}_{i+1}), \text{ for } i = 1 : p - 1 \tag{16}$$

with p is the number of nodes in the found path. Then, the terminal set can be computed as $\mathcal{X}_f = \lambda \mathcal{S}$. Our other concern is to fulfill the recursive feasibility. The MPC controller is recursively feasible if, starting from any feasible initial state $x_0 \in \tilde{\Pi}$, the MPC problem remains feasible for all time for any sequence of optimization variables. Related to the constraints shown in (12e), a reachability analysis is performed to determine whether the terminal set is reachable. One step backward reachability analysis can be performed as below,

$$\tilde{R}_j^i = A^{-1}(\tilde{R}_{j-1}^i \oplus (-BU)) \cap \tilde{\Pi}_i \tag{17}$$

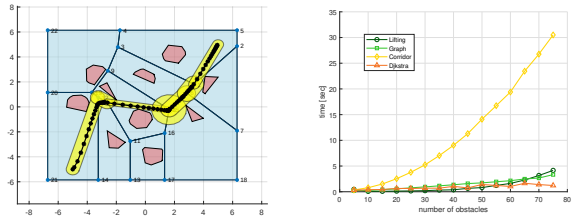
where the initial state can be selected as a terminal set, $\mathcal{X}_f$, and when the condition, $R^i = C\tilde{R}_j^i \subset \Pi_i$, is reached, the iteration count, j indicates the least number of predictions that is required to realize the recursive feasibility. The toolbox performs the BRS analysis by calling the "backwardsReachableSet()" method in the "mpc" class. To accelerate the BRS computation, BRS scaler in the "mpcsettings()" can be changed to re-discretize the system for this specific task.

From the application perspective, the mpc module in the toolbox is designed for the relay MPC algorithm that provides full trajectory inside of corridors that follow the path. The "mpc" object can be constructed as:

```
opt = mpcsettings();
mpcobj = mpc(opt.default())
```

where the system dynamics stored in default settings consist of a double integrator in the planar space. Furthermore, the relay MPC problem as in (11) and (12) can be computed for each corridor by calling "relayMPC" method.

```
    [flag, trajectory, controllerList] = mpcobj.
    relayMPC( path, PI, x0);
```



(a) Test in 2D space. (b) Time complexity.

Fig. 4: Numerical analysis on the right-hand side shows corridor construction (non-compulsory for the path planning) as most expensive task. The lifting procedure scales advantageously: up to 50 obstacles handled in the second range.

As the path, corridors, and initial states are the method's input, it returns the list of controllers navigating the agent through corridors, the full trajectory computed by the controllers with respect to the initial state, and the flag whether the optimization problem is solved successfully.

VI. Examples and Numerical Analysis

A. Example: Navigation in 2D cluttered environment

Let us consider agent dynamics defined as a discrete linear time-invariant (LTI) model,

$$x(k+1) = Ax(k) + Bu(k) \tag{18}$$
$$y(k) = Cx(k) + Du(k) \tag{19}$$

with $u \in \mathbb{U} \subset \mathbb{R}^m$ the control input and $x \in \mathbb{X} \subset \mathbb{R}^n$ and output $y \in \mathbb{Y} \subset \mathbb{R}^d$. Specially, we define state matrix, A input matrix, B, and output matrix, C for the discrete LTI as follows,

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; B = \begin{bmatrix} \Delta t^2/2 & 0 \\ 0 & \Delta t^2/2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{20}$$

where the disturbance matrix is taken into account as $D = 0^{2\times 2}$ and the MPC parameter $\mathrm{diag}(2, 2, 10, 10)$, $R = \mathbf{I}_2$, $u_{max} = -u_{min} = 2$. Then, dynamics are discretized with a $0.2s$ sampling period. The BRS scaler is selected as 10 for additional MPC parameters, and the fast terminal computation option and the terminal cost is set to 1. Regarding the convex lifting parameter, default options were chosen using "LiftOptions.linearDefault()". A random environment with 10 obstacles was generated, averaging 1m in size over a 30m space length. The "EdgeGraphBuilder()" was selected as the graph object with a corridor resolution set to 30. Upon executing the program "test2D.m", results are depicted in Fig. 4b. Space partitions, obstacles, corridors, and trajectories are visualized in light blue, red, yellow, and black, respectively.

B. Complexity Analysis

Complexity analysis can be seen in Fig 4b is performed to evaluate the computational effort of the present methods with respect to the varying number of obstacles. The analysis is conducted by generating four trials of random environments for each number of obstacles and averaging the results. The corridor computation involves solving the optimization (6). As one of the options is to calculate all possible corridor widths over the entire graph, its time complexity increases exponentially. This optional part can be streamlined by computing the widths only along the computed shortest path. On the other hand, the lifting-related optimization problem depends on the convexity condition of the PWA function. This relates to the number and distribution of obstacles. Their topology can be further exploited for lifting efficiency.

VII. Conclusions and Further Directions

The paper proposed a toolbox designated to handle problems related to the convex-lifting framework, including the recent development of convex-lifting in the context of path planning navigation. PNC-CL is an Apache-2.0 licensed open-source path-planning, navigation, and control software based on convex-lifting. It can be reached under `https://github.com/breakmit-0/PNC-CL` and utilized with respect to the provided instructions. The examples can also be found in the "ICARCV" tutorial folder.

The present toolbox can be used in conjunction with SLAM systems in order to get better knowledge of the environement and obstacles.

References

[1] S. Ceria, "A brief history of lift-and-project," *Annals of Operations Research*, vol. 149, no. 1, p. 57, 2007.

[2] H. D. Sherali and W. P. Adams, "A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems," *SIAM Journal on Discrete Mathematics*, vol. 3, no. 3, pp. 411–430, 1990.

[3] L. Lovász and A. Schrijver, "Cones of matrices and set-functions and 0–1 optimization," *SIAM journal on optimization*, vol. 1, no. 2, pp. 166–190, 1991.

[4] A. Berlinet and C. Thomas-Agnan, *Reproducing kernel Hilbert spaces in probability and statistics*. Springer, 2011.

[5] E. T. Maddalena, P. Scharnhorst, Y. Jiang, and C. N. Jones, "Kpc: Learning-based model predictive control with deterministic guarantees," in *Learning for Dynamics and Control*. PMLR, 2021, pp. 1015–1026.

[6] S. Yang, S. Olaru, and P. Rodriguez-Ayerbe, "Robust pwa control based on state space partitions," in *2023 European Control Conference (ECC)*. IEEE, 2023, pp. 1–6.

[7] ——, "On the inverse optimality of a class of pwa functions through liftings," in *2023 27th International Conference on System Theory, Control and Computing (ICSTCC)*, 2023, pp. 245–250.

[8] N. A. Nguyen, M. Gulan, S. Olaru, and P. Rodriguez-Ayerbe, "Convex lifting: Theory and control applications," *IEEE Transactions on Automatic Control*, vol. 63, no. 5, pp. 1243–1258, 2017.

[9] D. Ioan, S. Olaru, I. Prodan, F. Stoican, and S.-I. Niculescu, "From obstacle-based space partitioning to corridors and path planning. a convex lifting approach," *IEEE Control Systems Letters*, vol. 4, no. 1, pp. 79–84, 2019.

[10] K. Fukuda, "Cdd," 2022. [Online]. Available: https://people.inf.ethz.ch/fukudak/cdd_home/index.html

[11] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *2013 European control conference (ECC)*. IEEE, 2013, pp. 502–510.

[12] J. Löfberg, "Yalmip : A toolbox for modeling and optimization in matlab," in *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[13] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, no. 11, pp. 1747–1767, 1999.

[14] E. G. Gilbert and K. T. Tan, "Linear systems with state and control constraints: The theory and application of maximal output admissible sets," *IEEE Transactions on Automatic control*, vol. 36, no. 9, pp. 1008–1020, 1991.