MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

pycvxset: A Python package for convex set manipulation

Vinod, Abraham P.

TR2025-086 June 21, 2025

Abstract

This paper introduces pycvxset, a new Python package to manipulate and visualize convex sets. We support polytopes and ellipsoids, and provide user-friendly methods to perform a variety of set operations. For polytopes, pycvxset supports the standard halfspace/vertex representation as well as the constrained zonotope representation. The main advantage of constrained zonotope representations over standard halfs- pace/vertex representations is that constrained zonotopes admit closed-form expressions for several set operations. pycvxset uses CVXPY to solve various convex programs arising in set operations, and uses pycddlib to perform vertex-halfspace enu- meration. We demonstrate the use of pycvxset in analyzing and controlling dynamical systems in Python. pycvxset is available at https://github.com/merlresearch/pycvxset under the AGPL- 3.0-or-later license, along with documentation and examples.

American Control Conference (ACC) 2025

© 2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

pycvxset: A Python package for convex set manipulation

Abraham P. Vinod

***Abstract*— This paper introduces pycvxset, a new Python package to manipulate and visualize convex sets. We support polytopes and ellipsoids, and provide user-friendly methods to perform a variety of set operations. For polytopes, pycvxset supports the standard halfspace/vertex representation as well as the constrained zonotope representation. The main advantage of constrained zonotope representations over standard halfspace/vertex representations is that constrained zonotopes admit closed-form expressions for several set operations. pycvxset uses CVXPY to solve various convex programs arising in set operations, and uses pycddlib to perform vertex-halfspace enumeration. We demonstrate the use of pycvxset in analyzing and controlling dynamical systems in Python. pycvxset is available at https://github.com/merlresearch/pycvxset under the AGPL-3.0-or-later license, along with documentation and examples.**

I. Introduction

Set-based methods provide a formal framework to analyze and control dynamical systems. Such methods are often used in set propagation and reachability analysis where the goal is to characterize system states and controllers with desirable properties [1]–[4]. For example, in spacecraft rendezvous, set-based methods may be used to define a range of acceptable positions and velocities along the nominal spacecraft trajectory to guarantee safe abort [5]–[7]. See [1], [3], [8]–[10] for other applications of set-based control methods.

For linear systems, set-based methods yield practical implementations using efficient set representations like ellipsoids and polytopes. However, several set operations are not closed in ellipsoids [11], and certain set operations in the standard vertex/halfspace representation of polytopes require computationally expensive vertex-halfspace enumeration [3]. Recently, constrained zonotopes have been proposed for performing exact set operations on polytopes, since 1) they provide an equivalent representation of polytopes, and 2) they admit closed-form expressions for most operations [6]–[10].

Several open-source software toolboxes implement some or all of these set representations and their operations in various languages [11]–[18]. Together, these toolboxes have been instrumental in improving the access to set-based methods for reachability and trajectory optimization for the broader dynamical systems and control community. Compared to MATLAB tools [11], [16]–[18], existing tools in Python [12]–[15] focus mainly on halfspace/vertex-representation for polytopes and ignore other useful set representations like constrained zonotope and ellipsoids.

This paper *introduces* `pycvxset`, *a Python package to manipulate and visualize convex sets*. With `pycvxset`, we hope to bring the recent progress made in set representations especially constrained zonotopes [6]–[9] to Python. `pycvxset` extends `pytope` [13] to include ellipsoidal and constrained zonotopic set representations, broadens the capabilities of `Polytope` class including 3D plotting, and integrates with `CVXPY` for use in constrained control. `pycvxset` is extensively tested and documented for reliability and ease of use. See the project website https://merlresearch.github.io/pycvxset/ for more details.

A. P. Vinod is with Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA. Email: `abraham.p.vinod@ieee.org`.

II. Set representations and operations

A. Set representations

We consider three equivalent polytope representations:

$$\mathcal{P}(V) = \left\{ x \in \mathbb{R}^n \;\middle|\; \begin{array}{c} \exists \theta \in \mathbb{R}^N,\ x = V^\top \theta \\ 1^\top \theta = 1,\ \theta \geq 0 \end{array} \right\}, \quad (1)$$

$$\mathcal{P}(A, b, A_e, b_e) = \{x \in \mathbb{R}^n \mid Ax \leq b,\ A_e x = b_e\}, \quad (2)$$

$$\mathcal{P}(G, c, A_e, b_e) = \left\{ x \in \mathbb{R}^n \;\middle|\; \begin{array}{c} \exists \xi \in \mathbb{R}^N,\ x = G\xi + c, \\ \|\xi\|_\infty \leq 1,\ A_e x = b_e \end{array} \right\}, \quad (3)$$

with appropriate dimensions for V, A, b, A_e, b_e, G, c. Two set representations are said to be *equivalent* when the sets they represent contain each other. Here, (1) is the vertex representation (V-rep), (2) is the halfspace representation (H-rep), and (3) is the constrained zonotopic representation of a polytope. While the equivalence of (1) and (2) is well-known [3], their equivalence to a constrained zonotope representation was recently established in [8, Thm. 1]. The following sets are special cases of polytopes:

$$\mathcal{R}(l, u) = \{x \in \mathbb{R}^n \mid l \leq x \leq u\}, \quad (4)$$

$$\mathcal{R}(c, h) = \{x \in \mathbb{R}^n \mid \forall i \in \{1, 2, \ldots, n\},\ |x_i - c_i| \leq h_i\}, \quad (5)$$

$$\mathcal{Z}(G, c) = \{x \in \mathbb{R}^n \mid \exists \xi \in \mathbb{R}^N,\ x = G\xi + c,\ \|\xi\|_\infty \leq 1\}, \quad (6)$$

for finite vectors $l, u, c, h \in \mathbb{R}^n$ and appropriately dimensioned matrix G. Here, (4) and (5) represent axis-aligned rectangles, and (6) represents zonotopes. Throughout this paper, we refer to sets represented in the form of (1) and (2) as *polytopes* and those in the form of (3) as constrained zonotopes, even though they all represent the same set [8]. We refer to unbounded sets of the form (2) as *polyhedra*.

Finally, we consider bounded ellipsoidal sets $\mathcal{E}$ (7)–(9):

$$\mathcal{E}(Q, c) = \{x \in \mathbb{R}^n \mid (x - c)^\top Q^{-1} (x - c) \leq 1\}, \quad (7)$$

$$\mathcal{E}(G, c) = \{x \in \mathbb{R}^n \mid \exists \xi \in \mathbb{R}^{N_E},\ x = G\xi + c,\ \|\xi\|_2 \leq 1\}, \quad (8)$$

$$\mathcal{B}(c, r) = \{x \in \mathbb{R}^n \mid \|x - c\|_2 \leq r\}. \quad (9)$$

Any bounded, full-dimensional ellipsoid admits either of the representations — (7) with a positive definite matrix $Q \in \mathbb{R}^{n \times n}$ and (8) with $G \in \mathbb{R}^{n \times N_E}$ that has full-row rank [19].

Here, (7) and (8) satisfy $GG^\top = Q$. A bounded ellipsoid that is not full-dimensional may be represented using (8) with G that has linearly dependent rows, and (9) represents a ball with radius r.

B. Set operations

For any sets $\mathcal{T}, \mathcal{S} \subseteq \mathbb{R}^n$ and $\mathcal{W} \subseteq \mathbb{R}^m$, and a matrix $R \in \mathbb{R}^{m\times n}$, we define the set operations (affine map, Minkowski sum $\oplus$, intersection with inverse affine map $\cap_R$, and Pontryagin difference $\ominus$) as follows:

$$R\mathcal{T} \triangleq \{Ru : u \in \mathcal{T}\}, \tag{10a}$$

$$\mathcal{T} \oplus \mathcal{S} \triangleq \{u + v : u \in \mathcal{T},\ v \in \mathcal{S}\}, \tag{10b}$$

$$\mathcal{T} \cap_R \mathcal{W} \triangleq \{u \in \mathcal{T} : Ru \in \mathcal{W}\}, \tag{10c}$$

$$\mathcal{T} \ominus \mathcal{S} \triangleq \{u : \forall v \in \mathcal{S}, u + v \in \mathcal{T}\}. \tag{10d}$$

Since $\mathcal{T} \cap \mathcal{S} = \mathcal{T} \cap_{I_n} \mathcal{S}$, (10c) also includes the standard intersection. For any $x \in \mathbb{R}^n$, we use $\mathcal{T} + x$ and $\mathcal{T} - x$ to denote $\mathcal{T} \oplus \{x\}$ and $\mathcal{T} \oplus \{-x\}$ respectively for brevity. (10) also subsumes other set operations like orthogonal projection and inverse affine transformation (see Table I), and slicing (an intersection with an axis-aligned affine set).

The key advantage of constrained zonotopes over polytopes is that they admit closed form expressions for all set operations listed in (10) (except Pontryagin difference (10d)) [8], [9]. Recently, the authors have proposed a closed-form expression to inner-approximate the Pontryagin difference (10d) [6]. In contrast, polytopes must contend with computationally expensive vertex-halfspace enumeration when certain set operations are performed on a polytope in vertex/halfspace representation.

III. THE PYCVXSET PACKAGE

`pycvxset` provides three classes for representing convex sets (1)–(9): `Polytope`, `ConstrainedZonotope`, and `Ellipsoid`. `pycvxset` only supports bounded sets. In this section, we briefly discuss how to define, manipulate, and visualize these sets in Python using `pycvxset`.

A. Set definitions

1) Polytope: We define a polytope using the `Polytope` class in the following ways:
1) specifying (V) to define a polytope in V-rep (1),
2) specifying (A, b) or (A, b, Ae, be) to define a polytope in H-rep (2), and
3) specifying rectangles (l, u) (see (4)) or (c, h) (see (5)). We also provide methods to convert a polytope from V-rep (1) to H-rep (2) and vice versa using `pycddlib` and `scipy`.

The following code snippet creates a polytope in V-rep and 3-dimensional simplex in H-rep, prints the description of the polytope along with its vertices.

```
1 import numpy as np
2 from pycvxset import Polytope
3
4 V = [[-1, 0.5],[-1,1],[1,1],[1,-1],[0.5,-1]]
5 P1 = Polytope(V=V)
6 print("P1 is a", repr(P1))
7 A, b = -np.eye(3), np.zeros((3,))
8 Ae, be = [1, 1, 1], 1
9 P2 = Polytope(A=A, b=b, Ae=Ae, be=be)
10 print("P2 is a", P2)
11 print("Vertices of P2 are:\n", P2.V)
12 print("P2 is a", P2)
```

The above code snippet produces the following output:

```
P1 is a Polytope in R^2 in only V-rep
  In V-rep: 5 vertices
P2 is a Polytope in R^3 in only H-rep
Vertices of P2 are:
 [[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
P2 is a Polytope in R^3 in H-rep and V-rep
```

`pycvxset` supports exact conversion between (1) and (2) using `pycddlib`. For example, the call `P2.V` in Line 11 triggers a vertex enumeration internally as seen from the `print` statements for `P2`.

2) Constrained zonotope: We define a constrained zonotope (3) using the `ConstrainedZonotope` class in the following ways:
1) specifying a `Polytope` object as in (1) or (2),
2) specifying (c, G, Ae, be) as given in (3),
3) specifying rectangles (l, u) (4) or (c, h) (5), and
4) specifying (c, G) as given in (6) to define a zonotope.
We use [8, Thm. 1] to define a constrained zonotope $\mathcal{C}$ from $\mathcal{P}$ in (2). For $\mathcal{P}$ in (1), we first define a standard N-dimensional simplex in H-rep (2), use [8, Thm. 1] to compute the corresponding constrained zonotope, and then obtain $\mathcal{C}$ using the polytope vertices V and (10a).

The following code snippet creates a constrained zonotope from the polytope defined before as well as a box.

```
from pycvxset import ConstrainedZonotope

C1 = ConstrainedZonotope(polytope=P1)
print("C1 is a", repr(C1))
print("P1 is a", repr(P1))
C2 = ConstrainedZonotope(lb=[-1,-1], ub=[1,1])
print("C2 is a", repr(C2))
```

The above code snippet produces the following output:

```
C1 is a Constrained Zonotope in R^2
  with latent dimension 5 and 1 equality constraint
P1 is a Polytope in R^2 in only V-Rep
  In V-rep: 5 vertices
C2 is a Constrained Zonotope in R^2
  that is a zonotope with latent dimension 2
```

Note that `pycvxset` detects that `C2` is a zonotope.

`pycvxset` provide methods to generate polytopic approximations of constrained zonotopes (see Section III-B). Due to the computational effort involved [8], an exact conversion from (3) to (1) or (2) is not currently supported.

3) Ellipsoid: We define an ellipsoid using the `Ellipsoid` class in the following ways:
1) specifying (Q, c) as given in (7),
2) specifying (G, c) as given in (8), and
3) specifying (c, r) to define a ball (9).

`pycvxset` supports full-dimensional ellipsoids using (7) or (8), and degenerate ellipsoids using (8). The following code snippet creates two ellipsoids of the forms (7) and (8).

```
from pycvxset import Ellipsoid

E1 = Ellipsoid(c=[2,-1], Q=np.diag([1,4]))
print("E1 is an", E1)
E2 = Ellipsoid(c=[0,1,0], G=np.diag([1,2,3]))
print("E2 is an", E2)
```

The above code snippet produces the following output:

```
E1 is an Ellipsoid in R^2
E2 is an Ellipsoid in R^3
```

B. Visualizing polytopes and polytopic approximations

`pycvxset` can plot 2D and 3D polytopes using `matplotlib`. We can also plot polytopic approximations of constrained zonotopes and ellipsoids (see Fig. 1).

The following code snippet plots the sets in Fig. 1.

```
import matplotlib.pyplot as plt

plt.figure()
ax = plt.subplot(131, projection="3d")
P2.plot(ax=ax)   # Plot polytope
ax.view_init(elev=30, azim=-15)
ax.set_aspect("equal")
ax.set_title("Polytope")
ax = plt.subplot(132)   # Plot const. zonotope
C1.plot(ax=ax, vertex_args={"visible":True})
ax.set_aspect("equal")
ax.set_title("Constrained Zonotope")
ax = plt.subplot(133)   # Plot ellipsoid
E1.plot(ax=ax,patch_args={"facecolor":"pink"})
ax.set_aspect("equal")
ax.set_title("Ellipsoid")
plt.subplots_adjust(wspace=0.5)
```

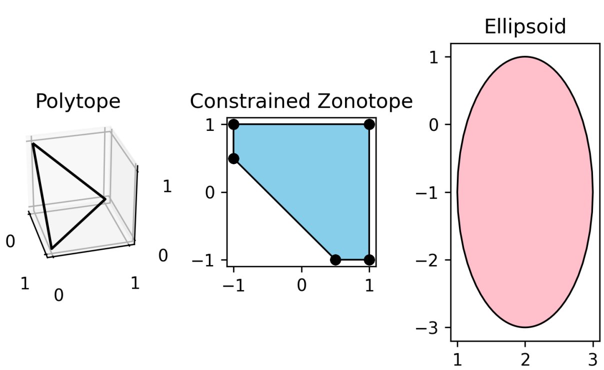

Fig. 1. Plotting various sets using `pycvxset`.

`pycvxset` provides flexibility in plotting either faces, vertices, or both, and provides identical methods for plotting irrespective of the set representation. By default, `pycvxset` plots inner-approximations of constrained zonotopes and ellipsoids, but outer-approximations may be plotted when required. For brevity, we will omit plotting commands in subsequent code snippets.

We compute polytopic (inner- and outer-) approximations for n-dimensional ellipsoids and constrained zonotopes using their support function and support vectors (see Table I). `pycvxset` auto-generates well-separated $2n + 2^n D$ direction vectors for any $D \in \mathbb{N}$ by solving the following optimization problem [20, Eq. (B.1)],

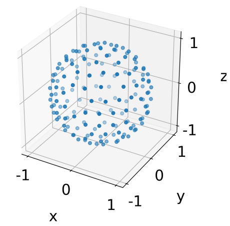

Fig. 2. Well-separated vectors on a 3-dimensional unit sphere using (11).

$$\begin{aligned} \text{max.} \quad & r \\ \text{s. t.} \quad & \|x_i - x_j\|_2 \geq r, && \forall 1 \leq i < j \leq D, \\ & \|x_i - e_j\|_2 \geq r, && \forall 1 \leq i \leq D,\ \forall 1 \leq j \leq n, \\ & 2x_i \geq r,\ 0.8 \leq \|x_i\| \leq 1, && \forall 1 \leq i \leq D. \end{aligned} \tag{11}$$

Here, the decision variables are vectors $x_i \in \mathbb{R}^n$ for $i \in \{1, 2, \ldots, D\}$ and a scalar r, and e_j denotes the standard axis vector in $\mathbb{R}^n$. (11) is a difference-of-convex program that aims to spread points x_i on the intersection of a unit sphere and the positive quadrant $\mathbb{R}^n_{\geq 0}$, which are subsequently reflected the axis planes to yield the direction vectors [20], [21]. (11) may be solved (approximately) via the well-known *convex-concave* procedure [22], and the approach is implemented in `pycvxset` as the method `spread_points_on_a_unit_sphere`. Fig. 2 shows the result of (11) for $n = 3$ and $D = 20$.

C. Set operations

Table I lists the set operations supported for each class. `pycvxset` provides identical methods for various set operations (when supported).

1) Involving another vector v and/or matrix M: For any set $\mathcal{X}$, we support affine transformation (M, v), inverse-affine transformation with an invertible map M, projecting a point v, checking if $v \in \mathcal{X}$, and computing the support function and vector of the set $\mathcal{X}$ along the direction v. We require inverse-affine map to have an invertible M to ensure that the pre-image of a bounded set under the affine map M is also bounded. These operations either have closed-form expressions (e.g., affine transformation of a constrained zonotope [8] or support function of an ellipsoid [11]) or require solving convex programs which we implement using `CVXPY` (e.g., checking $v \in \mathcal{X}$ for a polytope $\mathcal{X}$).

The following code computes the orthogonal projection on `P2` and the distance of a point $x = [1, 1, 1]$ from `P2` (Fig. 3).

```
projection, d = P2.project(x=[1, 1, 1], p=2)
```

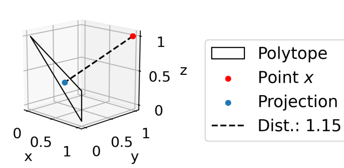

Fig. 3. Projection of a point on a polytope

TABLE I

PYCVXSET SET OPERATIONS ON A SET $\mathcal{X} \subset \mathbb{R}^n$. HERE, ✓ INDICATES EXACT IMPLEMENTATION, † INDICATES EXACT IMPLEMENTATION POSSIBLE AFTER CONVERTING POLYTOPE TO CONSTRAINEDZONOTOPE, AND ≈ INDICATES APPROXIMATE IMPLEMENTATION. ALL OPERATIONS FOR CONSTRAINED ZONOTOPES AND ELLIPSOIDS MAY BE APPROXIMATED, IF DESIRED, USING APPROPRIATE POLYTOPIC APPROXIMATIONS.

Operation	Expression	Polytope	Constrained zonotope	Ellipsoid
Set computations involving another vector v and/or matrix M				
Affine transformation (M, v) for $M \in \mathbb{R}^{m\times n}, v \in \mathbb{R}^m$	$M\mathcal{X} + v$	✓	✓	✓
Inverse-affine transformation $M \in \mathbb{R}^{n\times n}$, M is invertible	$\{x \mid Mx \in \mathcal{X}\}$	✓	✓	✓
Project a point $v \in \mathbb{R}^n$ on to $\mathcal{X}$ using $\|\cdot\|_p$, $p \in \{1, 2, \infty\}$	$\arg\min_{x\in\mathcal{X}} \|x - v\|_p$	✓	✓	✓
Containment of a point $v \in \mathbb{R}^n$ in $\mathcal{X}$	$v \in \mathcal{X}$	✓	✓	✓
Support function along a direction $v \in \mathbb{R}^n$	$\sup_{x\in\mathcal{X}} v^T x$	✓	✓	✓
Support vector along a direction $v \in \mathbb{R}^n$	$\arg\sup_{x\in\mathcal{X}} v^T x$	✓	✓	✓
Centering				
Chebyshev ball	$\sup_{\mathrm{Ball}(x,r)\subseteq\mathcal{X}}\ r$	✓	inner. ≈	✓
Maximum volume inscribed ellipsoid	$\sup_{\mathcal{E}(c,Q)\subseteq\mathcal{X}}\ \mathrm{Vol}(\mathcal{E})$	✓	inner. ≈	✓
Minimum volume circumscribed ellipsoid	$\inf_{\mathcal{E}(c,Q)\supseteq\mathcal{X}}\ \mathrm{Vol}(\mathcal{E})$	✓		✓
Minimum volume circumscribed rectangle	$\inf_{\mathrm{Rect}(l,u)\supseteq\mathcal{X}}\ \mathrm{Vol}(\mathrm{Rect})$	✓	✓	✓
Other set-specific manipulations/computations				
Interior point (Relative)	Compute $x \in \mathcal{X}$	✓	✓	✓
Orthogonal projection to $\mathbb{R}^m$	$\{x \mid \exists v \in \mathbb{R}^{n-m},\ [x; v] \in \mathcal{X}\}$	✓	✓	✓
Volume	$\mathrm{Vol}(\mathcal{X})$	✓	≈ $(n = 2)$	✓
Set computations involving another set $\mathcal{Y}$ ($\mathcal{Y} \subset \mathbb{R}^n$ unless specified otherwise)				
Intersection with a polytope $\mathcal{Y}$	$\mathcal{X} \cap \mathcal{Y}$	✓	✓	
Intersection with a polyhedron $\mathcal{Y}$		✓	✓	
Intersection with a constrained zonotope $\mathcal{Y}$		†	✓	
Intersection with $\mathcal{Y} \subset \mathbb{R}^m$ under inverse affine map $M \in \mathbb{R}^{m\times n}$		✓	✓	
Intersection with an affine set $\mathcal{Y}$ (includes slice operation)		✓	✓	✓
Minkowski sum with a polytope $\mathcal{Y}$	$\mathcal{X} \oplus \mathcal{Y}$	✓	✓	
Minkowski sum with a constrained zonotope $\mathcal{Y}$		†	✓	
Pontryagin difference with an ellipsoid $\mathcal{Y}$	$\mathcal{X} \ominus \mathcal{Y}$	✓	inner ≈	
Pontryagin difference with a zonotope $\mathcal{Y}$		✓	inner ≈	
Pontryagin difference with a polytope $\mathcal{Y}$		✓		
Containment of a polytope $\mathcal{Y}$	$\mathcal{Y} \subseteq \mathcal{X}$	✓	✓	✓
Containment of a constrained zonotope $\mathcal{Y}$		✓	✓	
Containment of an ellipsoid $\mathcal{Y}$		✓		✓

2) Centering: Centering methods provide a succinct, approximate representation of complex sets in the form of ellipsoids and rectangles [19, Ch. 8]. These methods solve convex programs for polytopes in V-rep/H-rep, and are available in closed-form for ellipsoids [11]. For constrained zonotopes, we provide approximations [7].

Fig. 4 illustrates centering and bounding sets for the polytope `P1` and the constrained zonotope `C1`, where we obtained identical Chebyshev ball, maximum volume inscribed ellipsoids, and minimum volume circumscribed rectangles.

3) Other set-specific manipulations/computations: We compute an interior point for each set using centering. For polytopes, we can also compute its centroid. We compute the orthogonal projection of a set using an appropriately defined affine map. We compute the orthogonal projection of a 3-dimensional unit ℓ_1-norm ball in the following code snippet (see Fig. 5). `pycvxset` counts dimensions from zero.

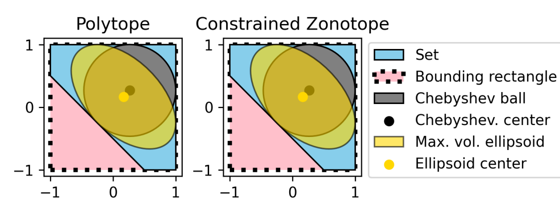

Fig. 4. Centering for a polytope `P1` and a constrained zonotope `C1`.

```
V = np.vstack((np.eye(3), -np.eye(3)))
l1ball = Polytope(V=V)
ball2D=l1ball.projection(project_away_dims=2)
```

We compute the volume of a full-dimensional polytope and an ellipsoid using `scipy` and closed-form expressions respectively. We approximate the volume of a constrained zonotope via grid-based sampling.

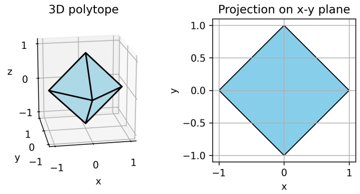

Fig. 5. Orthogonal projection of a unit ℓ_1-norm ball ($n = 3$).

4) Involving another set $\mathcal{Y}$: We provide exact implementations for intersection and Minkowski sum of polytopes and constrained zonotopes among themselves, and for Pontryagin difference of polytopes with any other sets [23]. The intersection and Minkowski sum of a constrained zonotope and a polytope returns a constrained zonotope. The Pontryagin difference of a constrained zonotope and an ellipsoid or a zonotope is inner-approximated with a constrained zonotope using least squares [6].

We implement an exact check for the containment of a polytope $\mathcal{Y}$ within a set $\mathcal{X}$ (not necessarily a polytope) by solving appropriate convex programs [19], where we check for the containment of all the vertices of $\mathcal{Y}$ in the set $\mathcal{X}$. We also implement an exact check for the containment of a set $\mathcal{Y}$ (not necessarily a polytope) within a polytope $\mathcal{X}$ using the support function [19]. We implement an exact check for the containment of an ellipsoid in another ellipsoid using semi-definite programming [19].

To implement exact check for $\mathcal{Y} \subseteq \mathcal{X}$ for two constrained zonotopes $\mathcal{X}, \mathcal{Y}$, we encode $x \in \mathcal{Y} \Rightarrow x \in \mathcal{X}$ as a bilinear program obtained using strong duality:

$$\begin{array}{rl} \text{minimize} & 1 + \alpha^\top (c_Y - G_X \xi_X - c_X) - \beta^\top b_{e,Y} \\ \text{subject to} & \|\xi_X\|_\infty \leq 1, \\ & A_{e,X}\xi = b_{e,X}, \\ & \|G_Y^\top \alpha + A_{e,Y}^\top \beta\|_1 \leq 1. \end{array} \tag{12}$$

with decision variables $\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}^{M_Y}, \xi_X \in \mathbb{R}^{N_X}$, and $\mathcal{Y} \subseteq \mathcal{X}$ if and only if the optimal value of (12) is non-negative. We solve (12) to optimality using CVXPY and GUROBI, and can also check for containment of polytopes within constrained zonotopes. These methods also enable checking for equality between polytopes and constrained zonotopes, as illustrated in the following code snippet.

```
 1 print("C1 is a", C1)
 2 print("P1 is a", P1)
 3 print("Are C1 and P1 equal?", C1 == P1)
 4 lb, ub, p, q = [-1, -1], [1, 1], [-1, -1], 0.5
 5 P1a = Polytope(lb=lb, ub=ub).
       intersection_with_halfspaces(A=p, b=q)
 6 C1a = ConstrainedZonotope(lb=lb, ub=ub).
       intersection_with_halfspaces(A=p, b=q)
 7 print("C1a is a", C1a)
 8 print("P1a is a", P1a)
 9 print("Are P1 and P1a equal?", P1a == P1a)
10 print("Are C1 and C1a equal?", C1 == C1a)
```

The above code snippet produces the following output:

```
C1 is a Constrained Zonotope in R^2
P1 is a Polytope in R^2 in H-Rep and V-Rep
Are C1 and P1 equal? True
C1a is a Constrained Zonotope in R^2
P1a is a Polytope in R^2 in only H-Rep
Are P1 and P1a equal? True
Are C1 and C1a equal? True
```

The equality of the sets C1 and P1 may also be visually confirmed in Fig. 4, where the polytopic inner-approximation of C1 computed by pycvxset for plotting is exact for this constrained zonotope instance. In contrast to the sets P1 and C1 defined in Sections III-A.1 and III-A.2, the sets P1a and C1a defined in Lines 5 and 6 in the above code snippet are defined by an intersection of a unit ℓ_∞-norm ball and an appropriate halfspace $\{x : p^\top x \leq q\}$. As expected, pycvxset declares all these sets to be equal, despite being different representations.

We support intersection of Polytope and ConstrainedZonotope objects with unbounded sets like affine sets and polyhedron, and the intersection of Ellipsoid with affine sets since these operations are also closed in Polytope, ConstrainedZonotope, and Ellipsoid respectively. We also implement slice using intersection with an appropriately-defined affine set.

We do not support intersection, Minkowski sum, and Pontryagin difference operations for ellipsoids natively in pycvxset since they are not closed operations for ellipsoids. However, all set operations discussed here are supported by Polytope. Consequently, any set operation that is not natively supported by pycvxset involving constrained zonotopes and ellipsoids may be approximated using their appropriate polytopic approximations (Section III-B).

D. Overloaded operators

We overload several Python operators to simplify the use of pycvxset. Table II summarizes how these operators interact with the sets in pycvxset. We interpret X + (-Y) and X - Y as $\mathcal{X} \oplus (-\mathcal{Y})$ and $\mathcal{X} \ominus \mathcal{Y}$ respectively.

When the comparison operators (<, <=, >, >=, **in**) are given with a set $\mathcal{X}$ and a vector y instead of a set $\mathcal{Y}$, pycvxset automatically switches to appropriate containment check with the vector y. Similarly, when the addition/subtraction operator is given a vector $y \in \mathbb{R}^n$, $\mathcal{X} + y$, $y + \mathcal{X}$, and $\mathcal{X} - y$ translates $\mathcal{X}$ by y, y, and $-y$ respectively.

TABLE II

PYTHON EXPRESSIONS SUPPORTED BY PYCVXSET INVOLVING SETS $\mathcal{X}, \mathcal{Y} \subset \mathbb{R}^n$, SCALAR $s \in \mathbb{R}$, VECTOR $v \in \mathbb{R}^n$, AND MATRIX M

Python expression	Interpretation
`M @ X`	Affine map with $M \in \mathbb{R}^{m \times n}$
`s * X`	Scaling: `(s * np.eye(X.dim)) @ X`
`-X`	Negation: `(-1) * X`
`X @ M`	Inverse affine map with invertible $M \in \mathbb{R}^{n \times n}$
`v < X, v <= X, v in X`	$v \in \mathcal{X}$
`X < Y, X <= Y, X in Y`	$\mathcal{X} \subseteq \mathcal{Y}$
`X > Y, X >= Y, Y in X`	$\mathcal{X} \supseteq \mathcal{Y}$
`X == Y`	Equality check: $\mathcal{X} \subseteq \mathcal{Y}$ and $\mathcal{Y} \subseteq \mathcal{X}$
`X + Y`	Minkowski sum of $\mathcal{X}$ with set $\mathcal{Y}$
`X - Y`	Pontryagin difference of $\mathcal{X}$ with set $\mathcal{Y}$
`X ** m`	Cartesian product with itself $m \in \mathbb{N}$ times

E. Solving relevant optimization problems

We use CVXPY to solve various optimization problems within pycvxset. We also provide methods to set up and solve convex programs with CVXPY involving sets constructed using pycvxset:

1) minimize to set up and solve optimization problems,

$$\begin{array}{ll}\text{minimize} & J(x),\\ \text{subject to} & x \in \mathcal{X},\end{array} \tag{13}$$

for any CVXPY-compatible cost function J, and

2) containment_constraints to obtain the CVXPY expressions that enforce the containment constraints $x \in \mathcal{X}$ as well as any necessary auxiliary variables.

Various methods in pycvxset like project, support use these methods to solve convex programs.

The user can specify the solver to use during set computations via the attributes cvxpy_args_lp, cvxpy_args_socp, and cvxpy_args_sdp associated with each object. These attributes are used when solving the various linear programs, second-order cone programs, and semi-definite programs respectively.

F. Installation and examples

We provide extensive documentation along with user-friendly examples for pycvxset at the project website: https://merlresearch.github.io/pycvxset/. The source code is available at https://github.com/merlresearch/pycvxset.

pycvxset is released under the AGPL-3.0-or-later license. We have tested pycvxset in Windows, Ubuntu, and MacOS, and for Python versions from 3.9 to 3.12. In future, we plan to register pycvxset to the Python Package Index as well.

IV. Reachability analysis using pycvxset

We now briefly discuss how pycvxset may be used to compute robust controllable (RC) sets [3, Defn. 10.18]. Consider a discrete-time linear time-invariant system with additive uncertainty,

$$x_{t+1} = Ax_t + Bu_t + Fw_t, \tag{14}$$

with state $x_t \in \mathbb{R}^n$, input $u_t \in \mathcal{U} \subset \mathbb{R}^m$, disturbance $w_t \in \mathcal{W} \subset \mathbb{R}^p$, and appropriate matrices A, B, F. We assume that the input set $\mathcal{U}$ and disturbance set $\mathcal{W}$ are convex and compact sets. Given a horizon $N \in \mathbb{N}$, a polytopic safe set $\mathcal{S} \subset \mathbb{R}^n$ and a polytopic target set $\mathcal{T} \subset \mathbb{R}^n$, a *$N$-step robust controllable set* is the set of initial states that can be robustly driven, through a time-varying control law, to the target set in N steps, while satisfying input and state constraints for all possible disturbances. Formally, we define the N-step RC set as $\mathcal{K}_0$ via the following set recursion for $t \in \{0, 1, \ldots, N-1\}$:

$$\mathcal{K}_t = \mathcal{S} \cap \left(A^{-1}((\mathcal{K}_{t+1} \ominus F\mathcal{W}) \oplus (-B\mathcal{U}))\right), \tag{15}$$

with $\mathcal{K}_N \triangleq \mathcal{T}$. We implement (15) in pycvxset with the following Python function get_rcs.

```
1 def get_rcs(S_U, S_W, S_S, S_T, A, B, F, N):
2     S_K = [None] * (N + 1)
3     S_K[-1], S_FW, S_BU = S_T, F@S_W, (-B)@S_U
4     for t in range(N - 1, -1, -1):
5         S_temp = (S_K[t+1] - S_FW) + S_BU
6         S_K[t]=S_S.intersection(S_temp @ A)
7     return S_K[0]
```

In get_rcs, we highlight variables denoting sets with a prefix S_ to distinguish from other variables — the horizon N and matrices A, B, F defined in (14). Here, S_U is $\mathcal{U}$, S_W is $\mathcal{W}$, S_T is $\mathcal{T}$, S_S is $\mathcal{S}$, and S_K is $\mathcal{K}$.

Lines 2 and 3 of get_rcs initialize the sets and pre-compute the affine-mapped sets in (15). Lines 5-6 implement the set recursion (15) using Table II. The returned set S_K[0] is a ConstrainedZonotope (or a Polytope) when sets S_U, S_T, and S_S are ConstrainedZonotope (or Polytope) objects. For a ConstrainedZonotope-based computation, the set S_W must be a zonotope or an ellipsoid [6], while the set S_W can be any set in pycvxset for the Polytope-based computation (see Table I).

V. Numerical examples

We provide two numerical examples to demonstrate various features of pycvxset. We also encourage readers to see [24] for more examples and additional details about pycvxset.

All computations were done on a standard laptop with 13th Gen Intel i7-1370P, 20 cores, 64 GB RAM using Python 3.9.

A. Reachability analysis for a double integrator

We use pycvxset to compute a 30-step RC set for a double integrator system model. A double integrator system can model an acceleration-controlled, mobile robot constrained to travel on a line. The corresponding RC set indicates the safe initial states that allow for subsequent satisfaction of state and input constraints. Using a sampling time of 0.1, we have (14) with,

$$A = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}, \ B = F = \begin{bmatrix} 0.005 \\ 0.1 \end{bmatrix}, \tag{16}$$

with two-dimensional state x_t denoting the position and velocity with one-dimensional input $u_t \in \mathcal{U} = [-1, 1]$ and disturbance $w_t \in \mathcal{W} = 0.4 \times \mathcal{U}$ denoting the controlled acceleration and the perturbation. We choose the safe set $\mathcal{S} = [-1, 1] \times [-0.5, 0.5]$, which serves as position and velocity bounds the robot must satisfy at all times. We choose the target set $\mathcal{T} = [-0.25, 0.25] \times [-0.1, 0.1]$, which requires the robot to have a terminal position (at time $t = 30$) within 0.25 m of the origin, and a terminal velocity magnitude of at most 0.1 m/s.

Fig. 6 shows the RC sets computed using get_rcs. Observe that the RC set computed using constrained zonotope is slightly smaller than the set computed using polytopes, due to the inner-approximation used in Pontryagin difference [6]. The overall computation time to generate and plot Fig. 6 was less than 3 seconds.

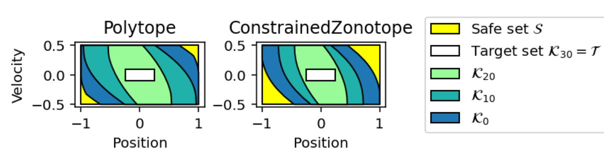

Fig. 6. 30-step RC sets for (16) using pycvxset.

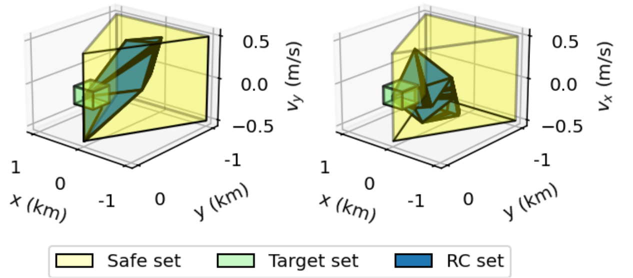

Fig. 7. Slices of the 4-dimensional 50-step RC set for a spacecraft rendezvous problem. (Left) Initial $v_x = 0$. (Right) Initial $v_y = 0$.

B. *Reach-avoid computation for spacecraft rendezvous*

We now demonstrate a practical application of `pycvxset` where `ConstrainedZonotope` class provides scalability and numerical stability over `Polytope` class for the computation of RC set. It also uses an ellipsoidal uncertainty set defined using `Ellipsoid` class.

We consider the problem of safe spacecraft rendezvous. For safety, it is essential to characterize the set of safe terminal configurations from which an approaching spacecraft (deputy) may wait for go/no-go for docking with another spacecraft (chief) [7], [20]. From each of these positions, the deputy must be able to proceed towards the chief for docking using bounded control authority while staying within a line-of-sight cone and satisfying velocity bounds at all times.

Dynamics: Assuming a circular orbit for the chief near the earth, the relative dynamics may be described by a four-dimensional linear system model, known as Hill-Clohessy-Wiltshire dynamics) to describe the position and velocity in relative `x`-`y` coordinates. We discretize the model in time using zero-order hold to obtain (14) with F set to a 4-dimensional identity matrix [7], [20], [21]. We assume that the thruster inputs $u_t \in \mathcal{U} = [-0.2, 0.2]^2$ N are held constant over the sampling time 30 seconds. We account for uncertainty in the rendezvous trajectory arising from potential actuator limitations of the spacecraft and model mismatch using an additive uncertainty $w_t \in \mathcal{W}_t = \texttt{Ellipsoid}(c = [0, 0, 0, 0], G = [10^{-5}, 10^{-5}, 10^{-4}, 10^{-4}]) \subset \mathbb{R}^4$ in the form (8) (units km and m/s).

Computation of RC set: We compute a 50-step RC set to navigate the deputy to a target set $\mathcal{T} = [-0.2, 0.2] \times [-0.2, 0] \times [-0.1, 0.1]^2$ (units km and m/s). Additionally, the deputy must remain inside a line-of-sight cone originating from the chief, $\mathcal{S} = \{x \in \mathbb{R}^4 : |x_1| \leq -x_2 \leq 1, |x_3| \leq 0.05, |x_4| \leq 0.05\}$ (units km and m/s). See [7], [20] for more details.

Fig. 7 shows the slices of the RC set computed using the `ConstrainedZonotope` class of `pycvxset`. We faced numerical issues when performing polytope-based computations of RC sets which may be attributed to the difficulties arising vertex-halfspace enumeration. The computation of the RC set using `ConstrainedZonotope` took about 20 seconds.

VI. Conclusion

This paper introduces `pycvxset`, an open-source Python package to manipulate and visualize convex sets in Python. Currently, `pycvxset` supports polytopic, ellipsoidal, and constrained zonotopic set representations. The packages facilitates the use of set-based methods to analyze and control dynamical systems in Python.

VII. Acknowledgements

We are grateful to Stefano Di Cairano and Kieran Parsons for their insightful feedback during the course of the development of this package.

References

[1] F. Blanchini and S. Miani, *Set-theoretic methods in control*. Springer, 2008.
[2] D. Bertsekas and I. Rhodes, “On the minimax reachability of target sets and target tubes,” *Automatica*, vol. 7, pp. 233–247, 1971.
[3] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge Univ. Press, 2017.
[4] M. Althoff, G. Frehse, and A. Girard, “Set propagation techniques for reachability analysis,” *Annual Rev. Ctrl., Rob., & Auto. Syst.*, vol. 4, no. 1, pp. 369–395, 2021.
[5] A. P. Vinod, A. Weiss, and S. Di Cairano, “Abort-safe spacecraft rendezvous under stochastic actuation and navigation uncertainty,” in *Proc. Conf. Dec. & Ctrl.*, 2021, pp. 6620–6625.
[6] ——, “Projection-free computation of robust controllable sets with constrained zonotopes,” *Automatica*, vol. 175, p. 112211, 2025.
[7] ——, “Inscribing and separating an ellipsoid and a constrained zonotope: Applications in stochastic control and centering,” in *Proc. Conf. Dec. & Ctrl*, 2024, pp. 8125–8131.
[8] J. Scott, D. Raimondo, G. Marseglia, and R. Braatz, “Constrained zonotopes: A new tool for set-based estimation and fault detection,” *Automatica*, vol. 69, pp. 126–136, 2016.
[9] V. Raghuraman and J. Koeln, “Set operations and order reductions for constrained zonotopes,” *Automatica*, vol. 139, p. 110204, 2022.
[10] L. Yang, H. Zhang, J. Jeannin, and N. Ozay, “Efficient backward reachability using the Minkowski difference of constrained zonotopes,” *IEEE Tran. Comp.-Aided Design Integrated. Circ. & Syst.*, vol. 41, no. 11, pp. 3969–3980, 2022.
[11] A. A. Kurzhanskiy and P. Varaiya, “Ellipsoidal toolbox (et),” in *Proc. Conf. Dec. & Ctrl.*, 2006, pp. 1498–1503, http://systemanalysisdpt-cmc-msu.github.io/ellipsoids/.
[12] “pycddlib,” https://pypi.org/project/pycddlib/.
[13] “pytope,” https://github.com/heirung/pytope.
[14] “polytope,” https://tulip-control.github.io/polytope/.
[15] “pypoman,” https://pypi.org/project/pypoman/.
[16] M. Althoff, “An introduction to CORA,” in *W. App. Verif. Cont. & Hyb. Syst.*, 2015, pp. 120–151, https://github.com/TUMcps/CORA.
[17] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, “Multi-parametric toolbox 3.0,” in *Proc. Euro. Ctrl. Conf.*, 2013, pp. 502–510, ttps://www.mpt3.org/.
[18] J. Koeln, T. J. Bird, J. Siefert, J. Ruths, H. C. Pangborn, and N. Jain, “zonoLAB: A MATLAB toolbox for set-based control systems analysis using hybrid zonotopes,” in *Proc. Amer. Ctrl. Conf.*, 2024, pp. 2513–2520, https://github.com/ESCL-at-UTD/zonoLAB.
[19] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge Univ. Press, 2004.
[20] J. Gleason, A. P. Vinod, and M. M. Oishi, “Lagrangian approximations for stochastic reachability of a target tube,” *Automatica*, vol. 128, 2021.
[21] A. P. Vinod, J. Gleason, and M. M. K. Oishi, “SReachTools: A MATLAB Stochastic Reachability Toolbox,” in *Proc. Hyb. Syst.: Comput. & Ctrl.*, Montreal, Canada, 2019, pp. 33 – 38, https://sreachtools.github.io.
[22] T. Lipp and S. Boyd, “Variations and extension of the convex–concave procedure,” *Opt. & Engg.*, vol. 17, pp. 263–287, 2016.
[23] I. Kolmanovsky and E. G. Gilbert, “Theory and computation of disturbance invariant sets for discrete-time linear systems,” *Mathematical problems in engineering*, vol. 4, no. 4, pp. 317–367, 1998.
[24] A. P. Vinod, “pycvxset: A python package for convex set manipulation,” *arXiv preprint arXiv:2410.11430*, 2024.