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

PYROBOCOP: Python-based Robotic Control &Optimization Package for Manipulation and Collision Avoidance

Raghunathan, Arvind; Jha, Devesh K.; Romeres, Diego

TR2024-087 July 02, 2024

Abstract

Contacts are central to most manipulation tasks as they provide additional dexterity to robots to perform challenging tasks. However, frictional contacts leads to complex complementarity constraints. Planning in the presence of contacts requires robust handling of these constraints to find feasible solutions. This paper presents PYROBOCOP which is a lightweight Python-based package for control and optimization of robotic systems described by nonlinear Differential Algebraic Equations (DAEs). In particular, the proposed optimization package can handle systems with contacts that are described by complementarity constraints. We also present a general framework for specifying obstacle avoidance constraints using complementarity constraints. The package performs direct transcription of the DAEs into a set of nonlinear equations by performing orthogonal collocation on finite elements. The resulting optimization problem belongs to the class of Mathematical Programs with Complementarity Constraints (MPCCs). MPCCs fail to satisfy commonly assumed constraint qualifications and require special handling of the complementarity constraints in order for NonLinear Program (NLP) solvers to solve them effectively. PYROBOCOP provides automatic reformulation of the complementarity constraints that enables NLP solvers to perform optimization of robotic systems. The package is interfaced with ADOL-C [1] for obtaining sparse derivatives by automatic differentiation and IPOPT [2] for performing optimization. We provide extensive numerical examples for various different robotic systems with collision avoidance as well as contact constraints represented using complementarity constraints. We provide comparisons with other open source optimization packages like CasADi and Pyomo. The code is open sourced and available at https://github.com/merlresearch/PyRoboCOP.

IEEE Transactions on Automation Science and Engineering 2024

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

PyRoboCOP: Python-based Robotic Control & Optimization Package for Manipulation and Collision Avoidance

Arvind U. Raghunathan[1], Devesh K. Jha[1]and Diego Romeres[1]

***Abstract*—Contacts are central to most manipulation tasks as they provide additional dexterity to robots to perform challenging tasks. However, frictional contacts leads to complex complementarity constraints. Planning in the presence of contacts requires robust handling of these constraints to find feasible solutions. This paper presents PyRoboCOP which is a lightweight Python-based package for control and optimization of robotic systems described by nonlinear Differential Algebraic Equations (DAEs). In particular, the proposed optimization package can handle systems with contacts that are described by complementarity constraints. We also present a general framework for specifying obstacle avoidance constraints using complementarity constraints. The package performs direct transcription of the DAEs into a set of nonlinear equations by performing orthogonal collocation on finite elements. The resulting optimization problem belongs to the class of Mathematical Programs with Complementarity Constraints (MPCCs). MPCCs fail to satisfy commonly assumed constraint qualifications and require special handling of the complementarity constraints in order for NonLinear Program (NLP) solvers to solve them effectively. PyRoboCOP provides automatic reformulation of the complementarity constraints that enables NLP solvers to perform optimization of robotic systems. The package is interfaced with ADOL-C [1] for obtaining sparse derivatives by automatic differentiation and IPOPT [2] for performing optimization. We provide extensive numerical examples for various different robotic systems with collision avoidance as well as contact constraints represented using complementarity constraints. We provide comparisons with other open source optimization packages like CasADi and Pyomo. The code is open sourced and available at https://github.com/merlresearch/PyRoboCOP.**

***Note to Practitioners*: PyRoboCOP is intended to be an easy-to-use software package written in Python which can be used for optimization, estimation and control for a large class of robotic systems. Including, in particular, contact-rich applications to deal with complex scenarios that arise when making and breaking contacts during a task. Typical problems that can be solved with our work are trajectory and control sequence optimization, parameter estimation. To make the proposed software package easier for practitioners, the paper provides access to the package and a large number of example problems. Furthermore, the package also provides a guide describing the details of all the methods a user might have to implement for their own system. Compared to some of the other packages, PyRoboCOP works with NumPy object arrays which is the native computing package in Python. We believe that this will make it much easier to learn and use compared to some of the other optimal control packages.**

I. Introduction

MOST robotic applications are characterized by presence of constrained and cluttered environments while dealing with challenging underlying phenomena like unilateral contacts, frictional contacts, impact and deformation [3]. These phenomena are very challenging to understand and represent mathematically. However, contacts are central to most of the manipulation problems and it is pivotal to model these phenomena. This has led to a lot of work towards development of fast and approximate models of multi-body dynamics [4]. However, to perform real-time optimization and control for contact-rich tasks, we must have tools to reason about the unique constraints imposed by contacts to discover optimal behavior.

Model-based control of contact-rich tasks could be facilitated by development of high-performance optimization algorithms that allow solution to mathematical programs for optimization in presence of contact constraints [5]. With this motivation, we present a Python-based robotic control and optimization package (called PyRoboCOP) that allows solution to optimal control problems for general dynamical systems with nonlinear constraints. The current paper and package only considers systems which can be represented by DAEs. Integration with physics engines is left as a future work as that requires additional development so as to expose the contact model specifications to PyRoboCOP for optimal performance. In particular, we employ collocation on finite elements to convert the infinite-dimensional optimal control problem to a finite-dimensional problem and is commonly referred to as direct transcription [6]. The direct transcription approach has been employed in a number of domains including trajectory optimization in aerospace industry [7], chemical process industry [8] among others. Another approach to performing optimization of dynamical systems is the shooting method whereby the DAE simulation engine is used to accurately integrate the dynamics and the continuity of the differential variables imposed using the optimization algorithm (see [6]). The advantage of the collocation method is that the inequality constraints are rendered into simple inequalities on discretized variables at the specific collocation point. On the other hand, multiple shooting methods require the computation of adjoint-based sensitivity in order to handle inequality constraints. We refer the interested reader to the book [6] and the recent paper [9] for a discussion on the different approaches for solving optimal control problems.

Contact-rich robotic manipulation tasks are mostly characterized by constraints imposed by contacts that could be modeled as complementarity constraints. Furthermore, manipulation in the presence of obstacles could become even more challenging due to the additional collision-avoidance constraints. Obtaining a feasible, let alone an optimal trajectory, can be challenging for such systems. An effective integration of the high-level trajectory planning in configura-

[1]All authors are with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139. Email– {raghunathan,jha,romeres}@merl.com

tion space with physics-based dynamics is necessary in order to obtain optimal performance of such robotic systems. To the best of author's knowledge, none of the existing Python-based open-source optimization packages can provide support for trajectory optimization with support for complementarity constraints that arise from contact-rich manipulation and an easy specification of obstacle avoidance constraints. Such optimization capability is, however, highly desirable to allow easy solution to optimization problems for a large-class of contact-rich robotic systems.

In this paper, we present PyRoboCOP – a lightweight but powerful Python-based package for control and optimization of robotic systems. A key contribution of our paper is that we present a novel complementarity-based formulation for modeling collision avoidance. Our formulation is differentiable even when the obstacles or objects are modeled as polytopes. The proposed formulation allows us to handle contact and collision avoidance in an unified manner. PyRoboCOP uses ADOL-C [1] and IPOPT [2] at its backend for automatic differentiation and optimization respectively. The main features of the package are:

- Direct transcription by orthogonal collocation on finite elements
- Contact modeling by complementarity constraints
- Obstacle avoidance modeling by complementarity constraints
- Support for minimum time problems
- Support for optimization over fixed mode sequence problems with unknown sequence time horizons
- Automatic differentiation for sparse derivatives

The features described above should convince the reader that PyRoboCOP addresses the identified gaps in existing software for optimization of robotic systems. By bringing together ADOL-C [1] and IPOPT [2] we believe that PyRoboCOP would be very useful for real-time model-based control of robotic systems. A preliminary version of the paper was presented as a conference publication [10]. Codes and instructions for installing and using PyRoboCOP could be found here https://github.com/merlresearch/PyRoboCOP.

Contributions. The main contributions of the paper are:

1) We present a Python-based package for optimization and control of a large class of robotic systems with contact and collision constraints.
2) We present a novel formulation for trajectory optimization in the presence of obstacles using complementarity constraints, thus allowing to solve for trajectory optimization of contact-rich systems in the presence of obstacles.
3) We evaluate our proposed package, PyRoboCOP , over a range of different dynamical systems (smooth as well as contact-rich) and also provide some comparison with SOTA optimization packages for performance and efficiency.

The rest of the paper is structured as follows. In Section II, we contrast PyRoboCOP to existing trajectory optimization techniques and present related works. Section III describes the dynamic optimization problem solved by PyRoboCOP and specifies the corresponding mathematical program obtained on collocation using finite elements. In Section IV, we describe several variants of discontinuous problems that could be solved by PyRoboCOP. Section V presents the user interface for PyRoboCOP. Finally, we show results on a range of different dynamical systems in Section VI. Conclusions and future work are summarized in Section VII.

II. Related Work

Our work is closely related to various optimization techniques proposed to solve contact-implicit trajectory optimization. Some related examples could be found in [11], [12], [13], [14], [15], [16], [17], [18]. In a more general setting, our work is related to trajectory optimization in the presence of non-differentiable constraints. These problems are common in systems with constraints like non-penetrability [19], minimum distance (e.g., in collision avoidance) [20], or in some cases robustness constraints [21].

Some of the existing open-source software for dynamic optimization are Optimica [22], ACADO Toolkit [23], TACO [24], pyomo.dae [25], Drake [26] and CasADi [27]. All of the cited software leverage automatic differentiation to provide the interfaced NLP solvers with first and second-order derivatives. However, these software do not provide any support for handling contact-based manipulation and obstacle-avoidance which are key requirements in robotic applications. More recently, some packages have been proposed to perform contact-rich tasks in robotics [28]. However, the solver proposed in [28] uses DDP-based [29] techniques which suffer from sub-optimality and difficulty in constraint satisfaction. Compared to most of the other techniques in open literature, the proposed optimization framework provides the following novelties and advantages.

1) Automatic transcription of DAEs to nonlinear equations using collocation on finite elements.
2) Automatic formulation of collision avoidance between pairs of objects with minimal user input.
3) Multiple formulations for handling complementarity constraints using NLP solvers robustly.

The optimization method presented in our work is most closely related to the direct trajectory optimization method for contact-rich systems earlier presented in [16], [17], [18]. The authors pose contact dynamics as a measure differential inclusion and employ an augmented Lagrangian to solve the resulting complementarity constrained optimization problem. [19] handle the complementarity constraint by relaxing to an inequality and solving using an active-set solver. In an analogous manner, other optimization packages like CasADi and Pyomo can also be extended for the solution to trajectory optimization in presence of complementarity constraints through a similar reformulation of the complementarity constraints. We provide an adaptive approach for relaxing the complementarity constraints. Further, we also provide a novel formulation for trajectory optimization in the presence of minimum distance constraints for collision avoidance. To the best of our knowledge, there is no other existing open-source, Python-based optimization toolbox that can handle constraints arising

due to frictional contact interaction and collision avoidance. Consequently, none of the existing trajectory optimization techniques and toolboxes could be used for solving frictional interaction in the presence of additional obstacles.

Our work is also an enabler for developing Nonlinear Model Predictive Control (NMPC) algorithms. Recent works [30], [31] have demonstrated the effectiveness of NMPC in solving collision avoidance problems. PyRoboCOP can be readily used to implement the MPC or NMPC frameworks using the existing functionality that we have made available. Indeed, in MPC and NMPC at each sampling instant, an optimization problem is solved to compute the control to be injected. PyRoboCOP can be utilized to solve that optimization problem at each sampling instant.

III. Problem Description

PyRoboCOP solves the dynamic optimization problem

$$\min_{x,y,u,p} \int_{t_0}^{t_f} c(x(t), y(t), u(t), p)dt + \phi(x(t_f), p) \tag{1a}$$

$$\text{s.t. } f(\dot{x}(t), x(t), y(t), u(t), p) = 0,\ x(t_0) = x_0 \tag{1b}$$

$$([y(t)]_{\sigma_{l,1}} - \nu_{l,1})([y(t)]_{\sigma_{l,2}} - \nu_{l,2}) = 0 \,\forall\, l \in \mathcal{L} \tag{1c}$$

$$\underline{x} \le x(t) \le \overline{x}, \underline{y} \le y(t) \le \overline{y}, \underline{u} \le u(t) \le \overline{u} \tag{1d}$$

where $x(t) \in \mathbb{R}^{n_x}$, $y(t) \in \mathbb{R}^{n_y}$, $u(t) \in \mathbb{R}^{n_u}$, $\dot{x}(t) \in \mathbb{R}^{n_x}$, $p \in \mathbb{R}^{n_p}$ are the differential, algebraic, control, time derivative of differential variables and time-invariant parameters respectively with the set of real numbers denoted by $\mathbb{R}$. The notation $[z]_k$ for a vector z refers to its k-th component. The function $\phi : \mathbb{R}^{n_x+n_p} \to \mathbb{R}$ represents Mayer-type objective function [32] term and is not a function of the entire trajectory. In addition, $\underline{x}, \overline{x}$, $\underline{y}, \overline{y}$, $\underline{u}, \overline{u}$ are the lower and upper bounds on the differential, algebraic and control variables. The initial condition for the differential variables is x_0. Constraints (1b)-(1c) are the Differential Algebraic Equations (DAEs) modeling the dynamics of the system with $f : \mathbb{R}^{2n_x+n_y+n_u} \to \mathbb{R}^{n_x+n_y-n_c}$ with $n_c = |\mathcal{L}|$. Each $l \in \mathcal{L}$ defines a pair of indices $\sigma_{l,1}, \sigma_{l,2} \in \{1, \ldots, n_y\}$ that specifies the complementarity constraint between the algebraic variables $[y(t)]_{\sigma_{l,1}}$ and $[y(t)]_{\sigma_{l,2}}$. In (1c) $\nu_{l,1}, \nu_{l,2}$ correspond to either the lower or upper bounds on the corresponding algebraic variables. For example, if they are set respectively to the lower and upper bounds of corresponding algebraic variables then (1c) in combination with the bounds (1d) model the complementarity constraint

$$0 \le [y(t) - \underline{y}]_{\sigma_{l,1}} \perp [\overline{y} - y(t)]_{\sigma_{l,2}} \ge 0.$$

The dynamic optimization problem in (1) is transcripted to a NonLinear Program (NLP) by orthogonal collocation on finite elements. The time interval $[t_0, t_f]$ is discretized into N_e finite elements of width h_i such that $\sum_{i=1}^{N_e} h_i = t_f - t_0$. Let $t_i = t_0 + \sum_{i' \le i} h_{i'}$ denote the ending time of the finite elements i. The differential, algebraic and control variables in each finite element $i \in \{1, \ldots, N_e\}$ are represented as Lagrange polynomials of degree (N_c+1), N_c and N_c respectively where N_c is the order of the collocation. Given $r_j \in (0, 1]$ for

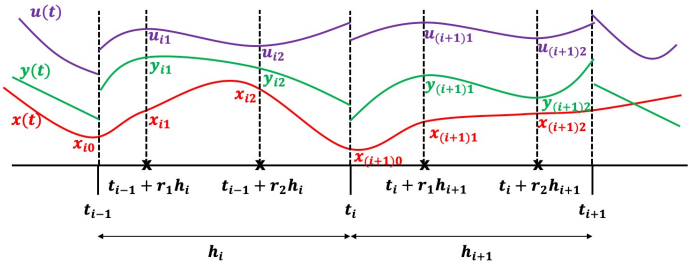


Fig. 1: Collocation on finite elements with $N_c = 2$. Note that the differential variables $(x(t))$ is continuous across finite elements while the algebraic $(y(t))$ and control $(u(t))$ variables are not continuous across the finite elements. The Lagrange polynomial representing $x(t)$ on finite element i is of degree N_c and is parameterized by x_{i0}, x_{i1}, x_{i2}. The Lagrange polynomial representing $y(t)(u(t))$ on finite element i is of degree $(N_c - 1)$ and is parameterized by $y_{i1}, y_{i2}(u_{i1}, u_{i2})$.

$j = 1, \ldots, N_c$ the differential, algebraic and control variables in the i-th finite element, i.e. $t \in [t_{i-1}, t_i]$, are represented as the following polynomials

$$x(t_{i-1} + \tau h_i) = \sum_{j=0}^{N_c} x_{ij} \Omega_j(\tau) \tag{2a}$$

$$y(t_{i-1} + \tau h_i) = \sum_{j=1}^{N_c} y_{ij} \Psi_j(\tau) \tag{2b}$$

$$u(t_{i-1} + \tau h_i) = \sum_{j=1}^{N_c} u_{ij} \Psi_j(\tau) \tag{2c}$$

$$\text{with } \Omega_j(\tau) = \prod_{k=0,\neq j}^{N_c} \frac{(\tau - r_k)}{(r_j - r_k)},\ \Psi_j(\tau) = \prod_{k=1,\neq j}^{N_c} \frac{(\tau - r_k)}{(r_j - r_k)}$$

where $r_0 = 0$, $\tau \in [0, 1]$ and x_{ij}, y_{ij}, u_{ij} denote the values of the differential, algebraic and control variables at time $(t_{i-i} + r_j h_i)$. The values of r_j are chosen as the roots of Legendre or shifted Radau polynomials of order N_c. Figure 1 provides a illustration of the polynomial representation of the differential, algebraic and control variables over the finite elements.

PyRoboCOP allows the user to employ a collocation method of order up to $N_c = 5$. The user has the option of using either Legendre or shifted Radau roots. In addition, the package also implements an explicit Euler scheme that can be useful in specifying discrete nonlinear systems as opposed DAEs. Such discrete nonlinear systems may arise from learned machine learning models from data, such as Gaussian Process (GP) and Neural Network models.

The NLP that results from applying the orthogonal collocation on finite elements to the dynamic optimization problem (1) is

$$\min \sum_{i=1}^{N_e} h_i c(x_{ij}, y_{ij}, u_{ij}) \widehat{\Psi}_j + \phi(x_f, p) \tag{3a}$$

$$\text{s.t. } f(\dot{x}_{ij}, x_{ij}, y_{ij}, u_{ij}) = 0,\ x_{10} = x_0 \tag{3b}$$

$$([y_{ij}]_{\sigma_{l,1}} - \nu_{l,1})([y_{ij}]_{\sigma_{l,2}} - \nu_{l,2}) = 0 \,\forall\, l \in \mathcal{L} \tag{3c}$$

$$\underline{x} \le x_{ij} \le \overline{x}, \underline{y} \le y_{ij} \le \overline{y}, \underline{u} \le u_{ij} \le \overline{u} \tag{3d}$$

$$\dot{x}_{ij} = h_i \sum_{k=0}^{N_c} x_{ik} \Omega'(\tau) \tag{3e}$$

$$x_{(i+1)0} = \sum_{j=0}^{N_c} x_{ij} \Omega_j(1) \tag{3f}$$

$$x_f = \sum_{j=0}^{N_c} x_{N_e j} \Omega_j(1) \tag{3g}$$

where the decision variables in the (3) are x_{ij}, $\dot{x}_{ij}$, y_{ij} and u_{ij}. The variables x_{ij} are indexed over $i \in \mathcal{N}_e (:= \{1, \ldots, N_e\})$ and $j \in \{0\} \cup \mathcal{N}_c (:= \{1, \ldots, N_c\})$, while the rest of the variables are indexed over $i \in \mathcal{N}_e$, $j \in \mathcal{N}_c$. The constraints (3b)-(3e) are imposed for $i \in \mathcal{N}_e$, $j \in \mathcal{N}_c$. The constraint in (3f) imposes the continuity of the differential variable across the finite elements and is imposed for $i \in \mathcal{N}_e \setminus \{N_e\}$. The constraint (3g) defines the state at the final time. The polynomial representation of x within a finite element i is used to relate $\dot{x}_{ij}$ to x_{ij} in (3e) where $\Omega'(\tau)$ is the derivative of $\Omega(\tau)$ w.r.t. τ. The notation $\widehat{\Psi}_j = \int_0^1 \Psi_j(\tau) d\tau$. If complementarity constraints are present then (3) is an instance of a Mathematical Program with Complementarity Constraints (MPCC).

MPCCs are well known to fail the standard Constraint Qualification (CQ) such as the Mangasarian Fromovitz CQ (MFCQ), see [33]. Hence, solution of MPCCs has warranted careful handling of the complementarity constraints when used in Interior Point Methods for NLP (IPM-NLP) using relaxation [34], [35] or penalty formulations [36]. In the case of active set methods, the robust solution of MPCC relies on special mechanism such as the elastic mode [37].

PyRoboCOP implements two possible relaxation schemes for complementarity constraints

$$\alpha_l([y_{ij}]_{\sigma_{l,1}} - \nu_{l,1})([y_{ij}]_{\sigma_{l,2}} - \nu_{l,2}) \leq \delta \, \forall \, l \in \mathcal{L} \tag{4a}$$

$$\sum_{l \in \mathcal{L}} \sum_{j=1}^{N_c} \alpha_l([y_{ij}]_{\sigma_{l,1}} - \nu_{l,1})([y_{ij}]_{\sigma_{l,2}} - \nu_{l,2}) \leq \delta n_c \tag{4b}$$

where $\alpha_l = 1$ if the involved bounds $(\nu_{l,1}, \nu_{l,2})$ are either both lower or both upper bounds. If one of the bounds is a lower bound and other is an upper bound then α_l is set to -1. Note that the choice of α_l ensures that the resulting product is nonnegative whenever (3d) are satisfied. The first approach relaxes each complementarity constraint by a positive parameter δ [34] while the second approach imposes the relaxation on the summation of all the complementarity constraints over a finite element i [37]. In addition, we also have flexibility to keep the δ fixed to a constant parameter through out the optimization or link this with the barrier parameter in IPM-NLP [34], [35].

When using time-stepping methods in the presence of complementarity constraints a lower integration error can only be obtained if the discontinuity is resolved accurately. Time-stepping methods for complementarity systems [38] do not employ a discontinuity resolution scheme and hence, the use of higher order collocation method is not meaningful. The approach in [39] addresses this problem by enforcing that the contacts do not change within a finite element. This allows them to employ higher order collocation. However, the resulting optimization problem tends to be more challenging and requires careful initialization for convergence. For the systems with complementarity constraints (1c), the current version of the package only allows the choice of $N_c = 1$ which corresponds to explicit or implicit Euler scheme. Higher values for N_c can be employed provided the lengths of the finite elements h_i are allowed to vary [40]. Such an approach renders the finite dimensional NLP quite difficult to solve unless a careful initialization of the iterates to guarantee convergence. We plan to consider extending PyRoboCOP for supporting collocation of higher orders in a future work.

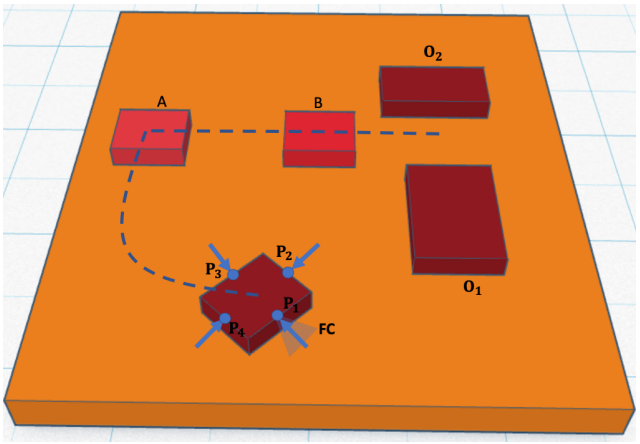


Fig. 2: Planar pushing in the presence of obstacles. Our proposed formulation in PyRoboCOP allows us to solve the collision avoidance with hybrid dynamics simultaneously using a novel formulation. The figure shows a possible solution to the given planning scenario. The figure shows a scenario where there are four possible points of contact for a pusher denoted as P_i, $i = 1, \ldots, 4$. An approximate friction cone is also shown at the point of contact.

IV. Trajectory Optimization Problems in Robotics

In this section, we describe three main instantiations of the mathematical programs that could be solved via the formulation described in Section III. These are problems with hybrid dynamics (e.g., systems with contacts using complementarity and mode enumeration) and problems with non-differentiable constraints like collision avoidance. It is noted that trajectory optimization problems for systems with smooth constraints are a subset of these problems and thus, are naturally solved by PyRoboCOP.

A. Manipulation Problems with Contacts

Robotic manipulation utilizes contacts to manipulate objects in its environment. Contacts appear in a lot of tasks in dexterous manipulation like pushing [41], [42], [43], prehensile manipulation [44], etc. A common manipulation primitive that we come across in robotics is pushing a body on a flat surface. Complementarity conditions could be used to model Coulomb friction between the pusher and the slider. Similarly, complementarity conditions could also appear because the pusher can exert force on a certain face of the slider (the object

being pushed) only on contact with that face, and thus the distance function between the contact surface and the pusher will be zero depending on which surface the contact happens (see Figure 2).

For sake of exposition, we will briefly describe a planar pushing model. For more detailed description of the pushing model, readers are referred to [45] and [46]. A schematic for a pusher-slider system is shown in Figure 3. The frictional interaction between the pusher and slider leads to a linear complementarity system which we describe next. The pusher interacts with the slider by exerting forces in the normal and tangential direction denoted by $f_{\overrightarrow{n}}$, $f_{\overrightarrow{t}}$ (as shown in Figure 3) as well as a torque τ about the center of the mass of the object. Assuming quasi-static interaction, the limit surface [47] defines an invertible relationship between applied wrench $\mathbf{w}$ and the twist of the slider $\mathbf{t}$. The applied wrench $\mathbf{w}$ causes the object to move in a perpendicular direction to the limit surface $\mathbf{H}(\mathbf{w})$. Consequently, the object twist in body frame is given by $\mathbf{t} = \nabla\mathbf{H}(\mathbf{w})$, where the applied wrench $\mathbf{w} = [f_{\overrightarrow{n}}, f_{\overrightarrow{t}}, \tau]$ could be written as $\mathbf{w} = \mathbf{J}^T(\overrightarrow{n} f_{\overrightarrow{n}} + \overrightarrow{t} f_{\overrightarrow{t}})$. For the contact configuration shown in Figure 3, the normal and tangential unit vectors are given by $\overrightarrow{n} = [1 \quad 0]^T$ and $\overrightarrow{t} = [0 \quad 1]^T$. The Jacobian $\mathbf{J}$ is given by $\mathbf{J} = \begin{bmatrix} 1 & 0 & -p_y \\ 0 & 1 & p_x \end{bmatrix}$.

The equation of motion of the pusher-slider system is then given by

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{R}\mathbf{t} \\ \dot{p}_y \end{bmatrix} \tag{5}$$

where $\mathbf{R}$ is the rotation matrix given by $\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Since the wrench applied on the system depends of the point of contact of pusher and slider, the state of the system is given by $\mathbf{x} = [x \quad y \quad \theta \quad p_y]^T$ and the input is given by $\mathbf{u} = [f_{\overrightarrow{n}} \quad f_{\overrightarrow{t}} \quad \dot{p}_y]^T$. The elements of the input vector must follow the laws of coulomb friction which can be expressed as complementarity conditions as follows:

$$\begin{aligned} 0 \leq \dot{p}_{y+} \perp (\mu_p f_{\overrightarrow{n}}(t) - f_{\overrightarrow{t}}(t)) \geq 0 \\ 0 \leq \dot{p}_{y-} \perp (\mu_p f_{\overrightarrow{n}}(t) + f_{\overrightarrow{t}}(t)) \geq 0 \end{aligned} \tag{6}$$

where $\dot{p}_y = \dot{p}_{y+} - \dot{p}_{y-}$ and the μ_p is the coefficient of friction between the pusher and the slider. The complementarity conditions in Eq. (6) mean that both $\dot{p}_{y+}$ and $\dot{p}_{y-}$ are non-negative and only one of them is non-zero at any time instant. Furthermore, $\dot{p}_y$ is non-zero only at the boundary of friction-cone. Consequently, the slipping velocity $\dot{p}_y$ cannot be chosen as an independent control input and is optimized while satisfying the conditions in Eq. (6).

B. Collision Avoidance

In this section, we present a novel formulation for the collision avoidance problem of rigid bodies. Path constraints arise in robotic systems due to their operation in cluttered environments. The constraints impose that the trajectories of robots do not collide with other obstacles which may be static or dynamic. For example, the pushing scenario shown in Figure 2 shows a frictional interaction scenario in the presence

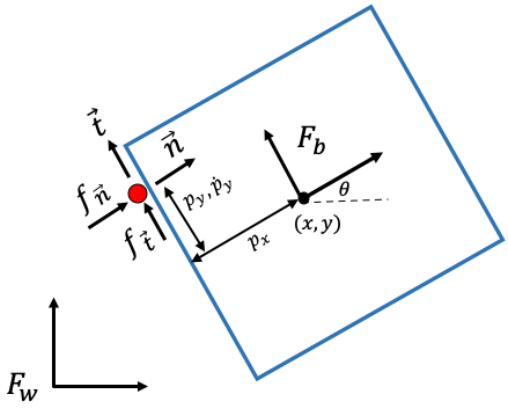


Fig. 3: A schematic of a planar pusher-slider system. State of the system is given by $[x, y, \theta, p_y]^T$ assuming that the pusher only comes in contact with the left edge as shown in the figure. The world frame and the body frame of reference are denoted by F_w and F_b respectively.

of obstacles. The goal in the example shown in Figure 2 is to move the object being pushed between the two obstacles O_1 and O_2.

We propose a novel formulation for collision avoidance using complementarity constraints. The distinguishing features of the formulation are that: (i) it is differentiable and allows for the use of NLP solvers and (ii) the treatment is identical for the case of robot-static obstacle and robot-robot collision avoidance.

We assume that the extent of the robot and obstacles are modeled as polytopes in 3-D and are specified by the set of vertices (see Figure 4). Let n_O denote the number of objects (including the robot and the obstacles). The polytope bounding the objects at time t are denoted by $\mathcal{O}_i(t)$ for $i = 1, \ldots, n_O$. We assume that user provides the matrix $V_i(x(t), y(t)) \in \mathbb{R}^{3 \times n_{vi}}$ with columns representing the coordinates of the n_{vi} vertices of the polytope $\mathcal{O}_i(t)$. The polytope $\mathcal{O}_i(t)$ is

$$\mathcal{O}_i(t) = \{q \,|\, q = V_i(x(t), y(t))\alpha, 1_{n_{vi}}^T \alpha = 1, \alpha \geq 0\} \tag{7}$$

where $1_{n_{vi}} \in \mathbb{R}^{n_{vi}}$ is a vector of all ones and $\alpha \in \mathbb{R}^{n_{vi}}$ are nonnegative vectors that generate the convex hull of the vertices. The dependence of the coordinates's vertices on the position, orientation of the object is modeled through the functional dependence of V_i on $x(t), y(t)$. Note that a static obstacle's vertex coordinates are independent of $x(t), y(t)$.

Consider two objects $i, j \in \{1, \ldots, n_O\}$. The distance between the objects at time t is

$$\min \|q_i - q_j\|^2 \text{ s.t. } q_i \in \mathcal{O}_i(t), q_j \in \mathcal{O}_j(t). \tag{8}$$

Suppose that the minimum separation required between the two objects is $\epsilon_{ij} > 0$. [48] proposed to model the distance constraint as

$$\|q_i - q_j\| \geq \epsilon_{ij} \text{ where } q_i, q_j \text{ solves (8).} \tag{9}$$

The formulation (9) is at best once differentiable when the minimizer of (8) is unique. The uniqueness requirement cannot

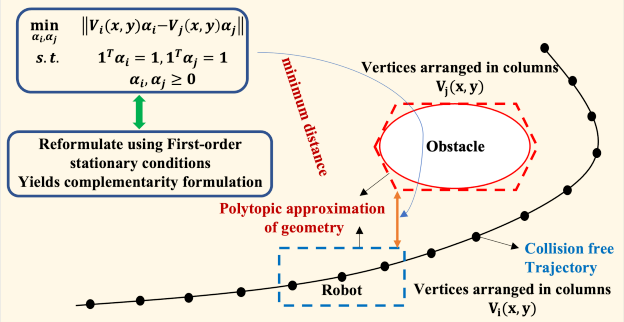


Fig. 4: Schematic showing the approximations needed to formulate an MPCC for the collision avoidance problem in PyRoboCOP. Using a polytopic representation for the objects in the environment allows us to represent first-order stationary conditions for minimum distance function as complementarity constraints.

be guaranteed when the objects are modeled as polytopes. As a consequence, the formulation in [48] cannot be directly provided to NLP solvers. Our formulation overcomes these drawbacks. The price to pay is that we need to solve a MPCC instead of a NLP. A schematic representing the underlying idea is shown in Figure 4.

The minimization problem (8) can be equivalently posed as

$$\min_{\alpha_{ij,i},\alpha_{ij,j}} \|V_i(x(t),y(t))\alpha_{ij,i} - V_j(x(t),y(t))\alpha_{ij,j}\|^2 \quad (10a)$$

$$\text{s.t. } 1^T_{n_{vi}}\alpha_{ij,i} = 1, 1^T_{n_{vj}}\alpha_{ij,j} = 1, \alpha_{ij,i}, \alpha_{ij,j} \geq 0 \quad (10b)$$

where $\alpha_{ij,i}$ and $\alpha_{ij,j}$ are variables denoting the convex combinations of the vertices of the polytope bounding the objects i, j (see Figure 4). At an optimal solution $\alpha^*_{ij,i}, \alpha^*_{ij,j}$ to (10) the points $V_i(x(t),y(t))\alpha^*_{ij,i}$ and $V_j(x(t),y(t))\alpha^*_{ij,j}$ are respectively the points in objects i, j that give the shortest distance between the objects. The optimization problem in (10) is a convex problem. Every first order stationary point of (10) is a minimizer [49]. This suggests imposing (10) through its first order stationary conditions

$$V_i(x(t),y(t))^T(V_i(x(t),y(t))\alpha_{ij,i} - V_j(x(t),y(t))\alpha_{ij,j}) + 1_{n_{vi}}\beta_i - \nu_i = 0 \quad (11a)$$

$$V_j(x(t),y(t))^T(V_j(x(t),y(t))\alpha_{ij,j} - V_i(x(t),y(t))\alpha_{ij,i}) + 1_{n_{vj}}\beta_j - \nu_j = 0 \quad (11b)$$

$$1^T_{n_{vi}}\alpha_{ij,i} = 1, 1^T_{n_{vj}}\alpha_{ij,j} = 1 \quad (11c)$$

$$0 \leq [\alpha_{ij,i}]_k \perp [\nu_i]_k \geq 0 \, \forall \, k = 1, \ldots, n_{vi} \quad (11d)$$

$$0 \leq [\alpha_{ij,i}]_k \perp [\nu_i]_k \geq 0 \, \forall \, k = 1, \ldots, n_{vj} \quad (11e)$$

where β_i, β_j are the multipliers for the equality constraints in (11c) and ν_i, ν_j are the multipliers for the nonnegative bounds on $\alpha_{ij,i}, \alpha_{ij,j}$. The separation requirement can be modeled as

$$\sqrt{\|V_i(x(t),y(t))\alpha_{ij,i} - V_j(x(t),y(t))\alpha_{ij,j}\|^2 + \epsilon^2} \geq \sqrt{\epsilon^2_{ij} + \epsilon^2} \quad (12)$$

The parameter $\epsilon > 0$ is a small constant that is included to render the constraint (12) differentiable everywhere. The separation requirements between the n_O objects are modeled through the constraints (11)-(12) for all $i < j$, with $i, j \in \{1, \ldots, n_O\}$.

The variables $\alpha_{ij,i}, \beta_i, \nu_i, \alpha_{ij,j}, \beta_j, \nu_j$ are in fact trajectories over time i.e. the variables are different for every time instant at which the collision avoidance constraint is imposed. The number of additional variables and constraints introduced for a pair of objects i, j is $(2n_{vi} + 2n_{vj} + 2)$. On applying the discretization, the total number of additional variables and constraints added to the MPCC is $N_e n_O(n_O - 1)(n_{vi} + n_{vj} + 1)$. The size of the MPCC scales quadratically in the number of objects and linearly in the number of vertices defining the objects. The constraints modeling collision avoidance are sparse. A sparsity preserving automatic differentiation algorithm and an NLP solver that can exploit such sparsity are critical to obtaining computational efficiency. We will touch upon in the design of our software next.

C. Optimizing Over Mode Sequences

Consider the complementarity constraint in (1c). A feasible point for (1c) requires either

$$[y(t)]_{\sigma_{l,1}} - \nu_{l,1} = 0 \text{ or } [y(t)]_{\sigma_{l,2}} - \nu_{l,2} = 0 \quad (13)$$

for each $l \in \mathcal{L}$. At every instant of time, the system can choose to enforce either one of the equalities in (13) for each $l \in \mathcal{L}$. This requirement reveals that (1c) embeds a disjunctive structure. This disjunctive nature of the system can be fully exposed through the notion of *modes*. At a particular time the mode of the system is defined as the set $m = \{(l, b(l)) \,|\, l \in \mathcal{L}, b(l) \in \{1, 2\}\}$ denoting which of the the two inequalities are satisfied as equality for each $l \in \mathcal{L}$. Note that set of of all possible modes $\mathcal{M}$ has cardinality $2^{|\mathcal{L}|}$. The formulation in (1) allows the system to be in any of modes in $\mathcal{M}$ at each instant of time assuming that the rest of the constraints can be satisfied. In certain robotic tasks, the sequence of modes is specified apriori but the time duration of the mode is not known (for example, consider a table-top manipulation scenario which can be performed using a known contact sequence, and thus known mode sequence). We can utilize PyRoboCOP to optimize the operation of such systems. We describe the continuous time-formulation for optimizing over mode sequences. The procedure described in III can be followed to convert the dynamic optimization problem to a nonlinear program.

Suppose for simplicity that the system is constrained to operate in two modes where the first mode is $m_1 = \{(l, 1) \,|\, l \in \mathcal{L}\}$ and the second mode is $m_2 = \{(l, 2) \,|\, l \in \mathcal{L}\}$. As mentioned earlier, the time duration for each of the modes needs to be determined as part of the optimization. Let T_1, T_2 denote the time duration for the two modes. Note that these are now parameters for optimization. Further, the duration of time in each mode is normalized to 1 instead of T_1, T_2 respectively and we employ a scaled time $\tilde{t} \in [0, 2]$. The system is in mode m_1 for $\tilde{t} \in [0, 1]$ and is in mode m_2 for $\tilde{t} \in [1, 2]$. The key advantage is that switching between modes can be precisely fixed in the scaled time ($\tilde{t} = 1$) coordinates which could not be done in absolute time coordinates. With this scaling of time,

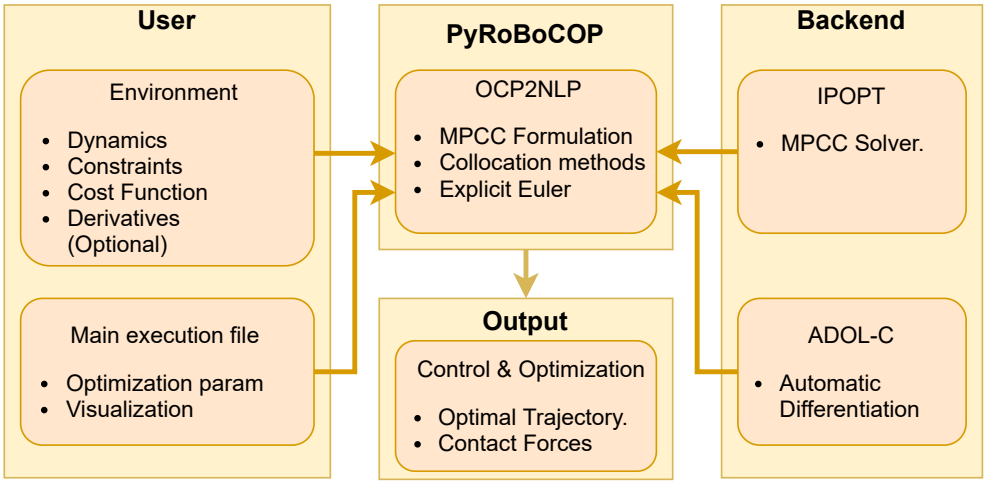


Fig. 5: Workflow in PyRoBoCOP. The dynamics provided by the user to create a MPCC which is then optimized using IPOPT and the gradients are evaluated using automatic differentiation via ADOL-C.

the variable $\dot{x}$ is equal to $\frac{1}{T_1}\tilde{\dot{x}}$ for $\tau \in [0,1]$ and is equal to $\frac{1}{T_2}\tilde{\dot{x}}$ for $\tau \in [1,2]$. The dynamics of the system (1b) can be recast as

$$f\left(\frac{1}{T_1}\tilde{\dot{x}}(\tilde{t}), y(\tilde{t}), u(\tilde{t}), p\right) = 0 \,\forall\, \tilde{t} \in [0,1] \tag{14a}$$

$$f\left(\frac{1}{T_2}\tilde{\dot{x}}(\tilde{t}), y(\tilde{t}), u(\tilde{t}), p\right) = 0 \,\forall\, \tilde{t} \in (1,2]. \tag{14b}$$

Further, the mode sequence is now realized by imposing time-varying bounds on the algebraic variables in the complementarity constraints, i.e.

$$[y(\tilde{t})]_{\sigma_{l,1}} \in \begin{cases} [\nu_{l,1}, \nu_{l,1}] & \text{for } \tilde{t} \in [0,1] \\ [[\underline{y}]_{\sigma_{l,1}}, [\overline{y}]_{\sigma_{l,1}}] & \text{for } \tilde{t} \in (1,2] \end{cases} \tag{15a}$$

$$[y(\tilde{t})]_{\sigma_{l,2}} \in \begin{cases} [[\underline{y}]_{\sigma_{l,2}}, [\overline{y}]_{\sigma_{l,2}}] & \text{for } \tilde{t} \in [0,1] \\ [\nu_{l,2}, \nu_{l,2}] & \text{for } \tilde{t} \in (1,2] \end{cases} \tag{15b}$$

The dynamic optimization problem over the mode sequence can be written as

$$\min_{x,y,u,p} \; T_1 \int_0^1 c(x(\tilde{t}), y(\tilde{t}), u(\tilde{t}), p) d\tilde{t} + T_2 \int_1^2 c(x(\tilde{t}), y(\tilde{t}), u(\tilde{t}), p) d\tilde{t} + \phi(x(2), p) \tag{16a}$$

$$\text{s.t. Eq. (14)}, \; x(0) = x_0 \tag{16b}$$

$$\text{Eq. (15)} \,\forall\, l \in \mathcal{L} \tag{16c}$$

$$\underline{x} \le x(\tilde{t}) \le \overline{x}, \underline{y} \le y(\tilde{t}) \le \overline{y}, \underline{u} \le u(\tilde{t}) \le \overline{u} \tag{16d}$$

The discretization can be applied to (16) to obtain a nonlinear program. Note that this formulation does not have complementarity constraints.

V. Software Description

Figure 5 provides a high-level summary of the flow of control in PyRoBoCOP. A user provided class specifies the dynamic optimization problem (1). This is also briefly described in Figure 5. The user needs to provide the equality constraints for the dynamical system. These constraints could include the dynamics information for the system, the bounds on the system state and inputs, and information about complementarity constraints, if any. Furthermore, a user needs to provide the objective function, and also has the option to provide derivative information (note the derivative information is optional). PyRoBoCOP expects the user provided class to implement the following methods in order to formulate an MPCC (or NLP) (also shown in Figure 5).

- `get_info`: Returns information on (1) including n_d, n_a, n_u, $|\mathcal{L}|$, n_p, N_e, h_i.
- `bounds`: Returns the lower and upper bounds on the variables $x(t)$, $\dot{x}(t)$, $y(t)$, $u(t)$ at a time instant t.
- `initialcondition`: Returns the initial conditions for the variables $x(t_0)$, i.e. values of the differential variables at initial time instant t_0.
- `initialpoint`: Returns the initial guess for the variables $x(t)$, $\dot{x}(t)$, $y(t)$, $u(t)$ at a time instant t. This initial guess is passed to the NLP solver.
- `objective`: Implements method to evaluate and return $c(x(t), y(t), u(t), p)$ at a time instant t.
- `constraint`: Implements method to evaluate and return $(f(x(t), \dot{x}(t), y(t), u(t), p)$ at a time instant t.

We provide a description of optional methods that are expected if certain specified conditions are satisfied.

- `bounds_finaltime`: Returns the bounds on the variables $x(t_f)$ at the final time. This method allows to specify a final time condition on a subset or all of the differential variables.
- `bounds_params`: Returns information on lower and upper bounds on the parameters p. This method must be implemented if $n_p > 0$.
- `initialpoint_params`: Returns the initial guess for the parameters p. This method must be implemented if $n_p > 0$. This initial guess is passed the NLP solver.
- `get_complementarity_info`: Returns information

on the complementarity constraints in (1) i.e. $\mathcal{L}$ and also information on whether the lower or upper bound is involved in the complementarity constraint. This method must be implemented if $\mathcal{L} \neq \emptyset$.
- `objective_mayer`: Implements method to evaluate and return $\phi(x(t_f), p)$.
- `get_objects_info`: Returns the information on number of objects n_O, flags to indicate if these obstacles are static or dynamic and the number of vertices n_{vi} for the polytope bounding the objects.
- `get_object_vertices`: Implements and returns the matrix $V_i(x(t), y(t)) \in \mathbb{R}^{3\times n_{vi}}$ representing the vertices of the polytope bounding the objects. This method is called only when `get_objects_info` is implemented and $n_O > 0$.

Note that the user is not required to implement the collision avoidance constraints (11)-(12). The user only provides the matrix $V_i(x(t), y(t))$ which models the dependence of the vertices of the bounding polytope on the differential and algebraic variables in (1). PyRoboCOP defines a new class that wraps around the user-provided class to provide a dynamic optimization problem that is of the form in (1). The new class includes additional variables and constraints modeling the collision avoidance constraints.

PyRoboCOP is interfaced with ADOL-C [1] to compute derivatives (see the Backend block in Figure 5). Note that the ADOL-C can also provide the sparsity pattern of the constraint jacobian and hessian of the Lagrangian. As mentioned earlier, the exploitation of sparsity in computations of the NLP is critical to solve large problems. To provide derivatives PyRoboCOP used ADOL-C to set up tapes [1] for evaluating: (i) the objective (3a), (ii) constraints including the DAE (3b) and a reformulation of (3c), and (iii) the Hessian of the Lagrangian of the NLP (3). The set-up of the tape is done prior to passing control to the NLP solver. The advantage of this approach is that evaluation of (3a), (3b) are now C-function calls instead of Python-function calls. This considerably reduced the time spent in function evaluations for the NLP solver. As shown in Figure 5, PyRoboCOP uses IPOPT as the optimization solver.

The user has the ability to specify a number of parameters that determine the type of collocations method, complementarity relaxation that PyRoboCOP used for discretization of the optimal control problem and reformulation for solution by an optimization solver. We list these parameters and provide a brief description. For a comprehensive description of the usage and an example, please refer to the software documentation[1]

- Number of collocation points (N_c): User specifies the number of collocation points to be used in each finite element $i \in \mathcal{N}_e$. PyRoboCOP supports up to order 5.
- Roots for the collocation points (r_k): User specifies the roots to be used for determining the locations of the collocation points in each finite element $i \in \mathcal{N}_e$. The available options are "legendre" and "radau".
- Choice of complementarity relaxation and adpative relaxations: The user has flexibility in specifying how the complementarity constraints are solved. The choices are: (i) (4a) with δ fixed, (ii) (4b) with δ fixed, (iii) (4a) with δ set equal to the interior point barrier parameter, (iv) (4b) with δ set equal to the interior point barrier parameter, and (v) the objective function is appended with complementarity terms as $\sum_{i=1}^{N_e} \sum_{l\in\mathcal{L}} \sum_{j=1}^{N_c} \alpha_l([y_{ij}]_{\sigma_{l,1}} - \nu_{l,1})([y_{ij}]_{\sigma_{l,2}} - \nu_{l,2})$. The convergence behavior of formulations can be quite different and we provide these implementations so the user can choose one that works best for the problem at hand.
- Choice of user-providing derivatives: The user can choose to provide the first and second derivatives or select to use ADOL-C for providing the derivatives automatically.

Codes and instructions for installing and using PyRoboCOP could be found here https://github.com/merlresearch/PyRoboCOP.

[1]https://github.com/merlresearch/PyRoboCOP/blob/main/PyRoBoCOP_Software_Package.pdf

VI. Numerical results

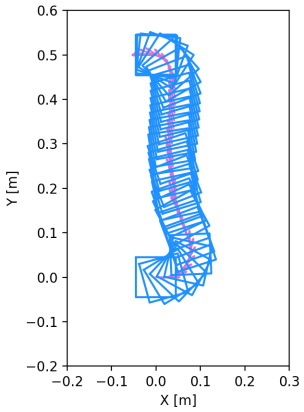
In this section, we test PyRoboCOP in several robotic simulations providing solutions to trajectory optimization problems including several systems with complementarity constraints. In all these examples, we do not provide a feasible initialization, and the performance of PyRoboCOP would be enhanced with a better initialization.

To foster reproducibility, the source code for each of the following examples is available at https://github.com/merlresearch/PyRoboCOP.

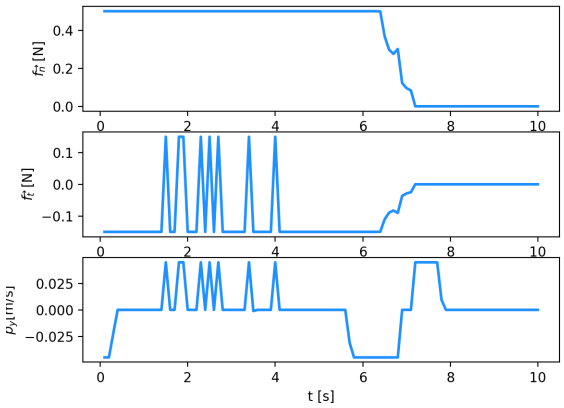
A. Planar Pushing

In this section, we show some results for planar pushing without any obstacles. The model for planar pushing was earlier presented in Section IV-A (see Eqs (5) and (6)). The complementarity constraints are used to represent slipping or sticking contact between the slider and the pusher. Two pushing trajectories with different goal configurations from the same initial state are shown in Figure 6. In both these examples, the initial pose of the slider is $\mathbf{x}_{\texttt{init}} = (0, 0, 0)$ and the desired goal pose of the slider is $\mathbf{x_g} = (0, 0.5, \pi)$ and $(0, 0, \pi)$. The initial point of contact between the pusher and the slider is $p_y = 0$. For all these examples, the maximum normal force is set to 0.5 N and the coefficient of friction is $\mu_p = 0.3$. The corresponding control trajectory shows the sequence of forces f_n and f_t used by the slider to obtain the desired trajectory. The plot of $\dot{p}_y$ shows the sequence of sticking and slipping contact as found by PyRoboCOP and thus this also decides the contact point between the pusher and the slider. Note that the pusher maintains sticking contact with slider whenever $\dot{p}_y = 0$, and slipping contact otherwise. In both these examples, the objective function is a function of the target state and the control inputs. It is noted that the modes for the pushing problem described here is sticking and slipping contact between the pusher and the slider.

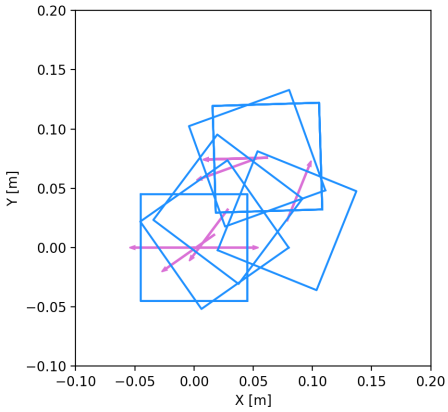
To demonstrate the robustness of our approach to change in goal configurations, we considered 100 different goal states



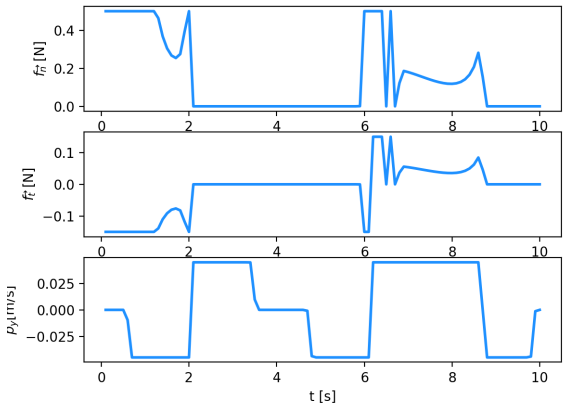
(a) Pushing Sequence for $\mathbf{x_g} = (0, 0.5, \pi)$



(b) Optimal Controls



(c) Pushing sequence for $\mathbf{x_g} = (0, 0, \pi)$



(d) Optimal Controls

Fig. 6: Optimal pushing sequences and control inputs obtained by solving the MPCC for two different goal conditions. The switching sequence between sticking and slipping contact formation could be visualized by the trajectory of $\dot{p}_y$. The pusher maintains a sticking contact with the slider when $\dot{p}_y = 0$. For clarity, we show very few frames of the pushing sequence in the second example in plot 6c.

given by $\mathbf{x_g} = (0, 0.5{*}i/10, \pi{*}j/10)$ for $i, j = 1, \ldots, 10$. We ran the optimization for each of the goal states without any modification to the initialization of the optimization problem. Of these 100 problems, IPOPT converged to a solution in 96 instances. Thus, PyRoboCOP shows the ability to generate optimal trajectories for a range of goal states. We must stress that MPCCs are challenging to solve for nonlinear programming algorithms. As noted, we have attempted to solve the 100 problems without modifying the initialization. A more careful initialization that is specific to the instance can provide convergence in all instances.

B. Car Parking Example with Obstacle Avoidance

To show collision avoidance, we show a parking scenario which has been previously used to show the effectiveness of several optimization-based collision-avoidance methods [20]. The dynamics of the car is described in [20]. The state of the car is defined by a 4-dimensional vector $\mathbf{x} = [x, y, \theta, v]^T$, where x, y is the center of the rear axis, θ is the heading angle and v is the longitudinal velocity of the car. The initial state of the car was chosen to be $\mathbf{x}_{\texttt{init}} = (1, 4, 0, 0)$ and the desired state was chosen to be $\mathbf{x}_g = (2, 2.5, \pi/2, 0)$. The resulting optimal solution from PyRoboCOP is shown in Figure 7. As described earlier in Section IV-B, the two static obstacles are specified by providing the vertex set for them.

C. Assembly of Belt Drive Unit

An example of a complex manipulation problem that involves contacts, elastic objects and collision avoidance is provided by the Belt Drive Unit system. This assembly challenge was presented as one of the most challenging competition in the World Robot Summit 2018[2] [50]. The real world system is represented in Figure 8 where the objective of the manipulation problem is to wrap the belt, held by a robotic manipulator around the two pulleys. The elastic belt is modeled through a 3D keypoint representation. The hybrid behavior of the model generated by the contacts between the belt and the pulleys and the elastic properties of the belt is captured by the complementarity constraints.

[2]https://worldrobotsummit.org/en/about/

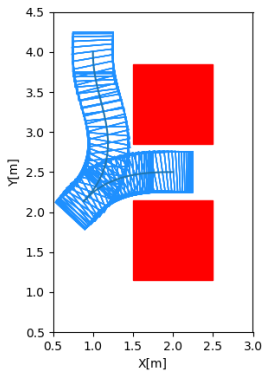


Fig. 7: Motion Planning in the presence of obstacles using the proposed obstacle avoidance method using complementarity constraints.

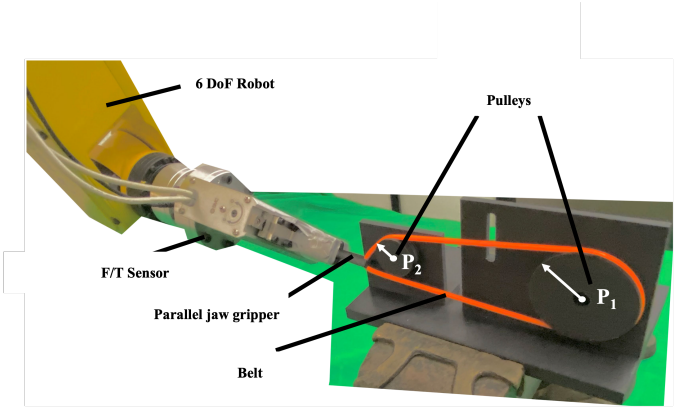


Fig. 8: Real Setup of the Belt Drive Unit system.

The full manipulation task has been divided into two subtasks as shown in Figure 9. The goals of the first and second subtask are to wrap the belt around the first and second pulley, respectively. The elastic belt was modeled by

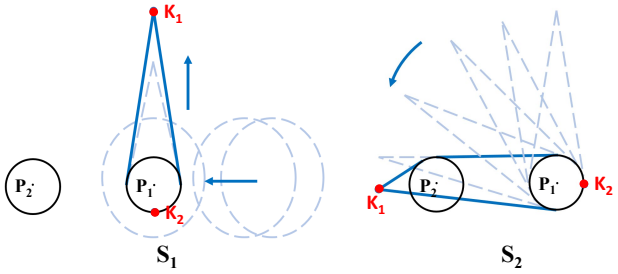


Fig. 9: Visualization of the two subtasks decomposition, S_1 and S_2. P_1 and P_2 are two pulleys. The blue lines represent the belt gripped by a robot at keypoint K_1, and K_2 is the lower keypoint. S_1: The belt wraps around the first pulley P_1 and it is stretched. S_2: The belt rotates around the first pulley and it is assembled onto the second pulley P_2.

two points, called keypoints: the upper keypoint K_1 and the lower keypoint K_2. K_1 is placed where the end-effector grasps the belt, and K_2 is the lowest point when the belt is falling under gravity because being lifted up by K_1. See Figure 9 for a schematic visualization. The dynamics and the physical constraints were modeled for this reduced representation. In particular, the complementarity constraints at the two keypoints model the elastic forces to describe when the belt is stretched or loose and the contact forces to describe when the belt is in contact with the pulleys. With these constraints we formulated two trajectory optimization problems one for each of the two subtasks as a MPCC of the form in (1). More

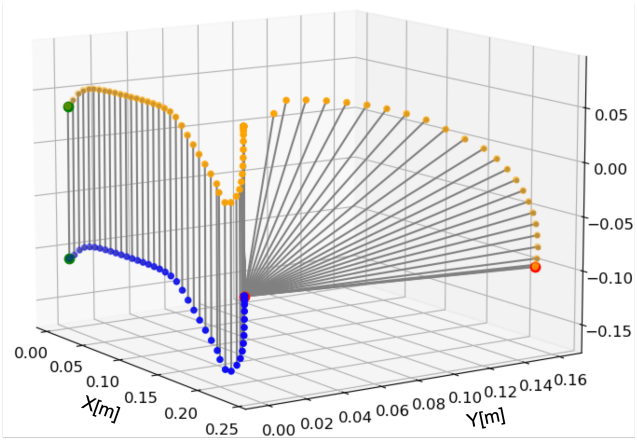
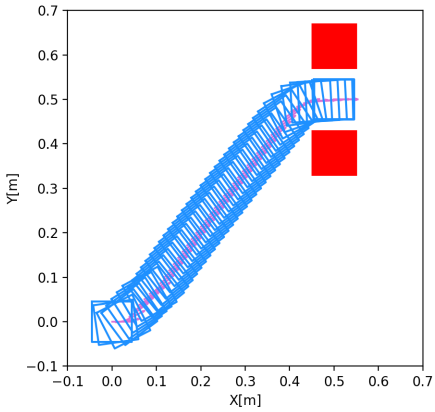


Fig. 10: The optimal trajectory to assemble the belt is shown. The orange points represent the trajectory of the upper keypoint, K_1, and the blue points represent the lower keypoint, K_2, which together represent the model of the belt. The green and red points are the starting and the final points, respectively. The grey lines are virtual connections between K_1 and K_2 for illustration only. The belt approaches the first pulley (not shown) then there is a movement downwards to hook the pulley with the lower keypoint from below during subtask 1. The lower keypoint is then hooked onto the pulley and will not move. Then, the higher keypoint, K_2, moves toward the second pulley (not shown) stretching the belt and wraps around the pulley during subtask 2.

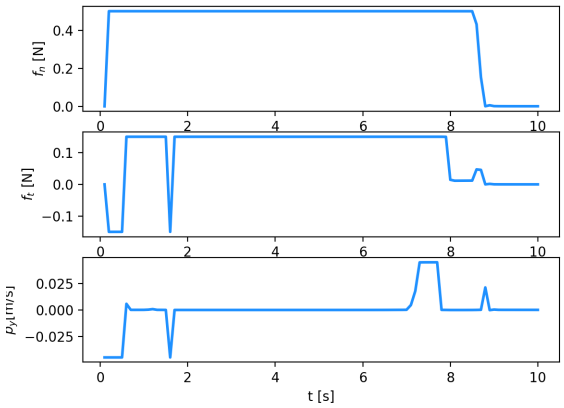
details on the modeling assumptions, the division into the two subtasks, the exact formulation including the explanation of the dynamics and complementarity constraints can be found in our previous paper [51]. In Figure 10 we report successful trajectories computed by PyRoboCOP to assemble the belt drive unit combining the optimal trajectories obtained in the two subtasks. The optimal trajectory was implemented on the real system with a tracking controller, see [51] for further details.

D. Planar Pushing With Obstacles

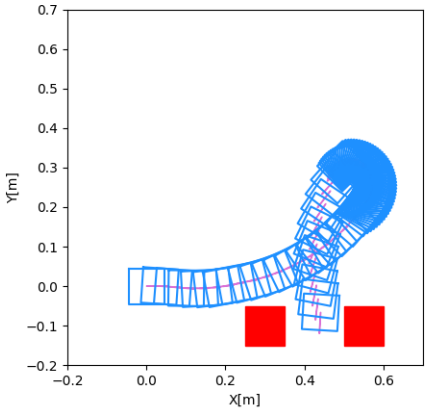
In this section, we show the solution to some planar pushing scenarios in the presence of obstacles and show that our proposed method can handle complementarity constraints as well as obstacle avoidance constraints simultaneously. We demonstrate our approach on two different pushing scenarios with same initial condition for the slider but different location of the obstacles and different goal states for the slider. In



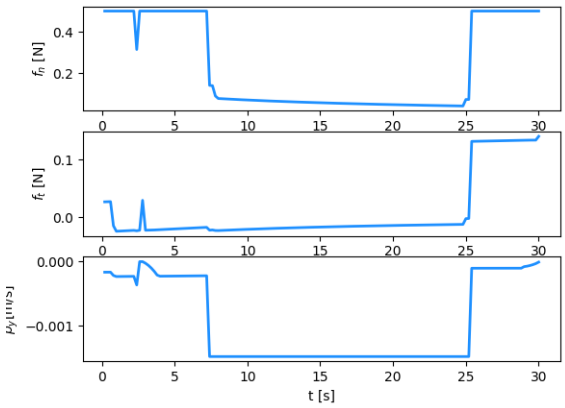
(a) Pushing sequence for initial position $(0, 0, 0)$ and desired goal $(0.5, 0.5, 0)$



(b) Optimal Controls obtained for Example 11a



(c) Pushing sequence for initial position $(0, 0, 0)$ and desired goal $(0.45, -0.1, 3\pi/2.)$



(d) Optimal Controls obtained for Example 11c.

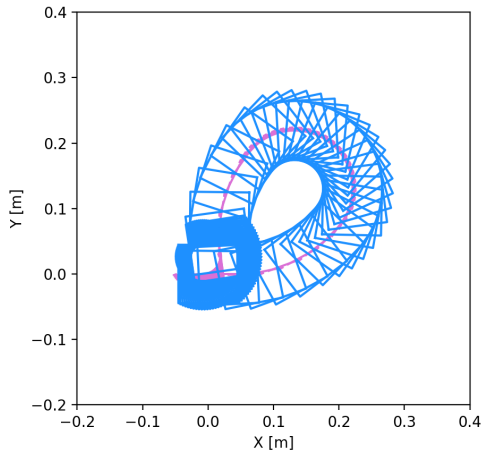
Fig. 11: Planar pushing in the presence of obstacles. Our proposed formulation in PyRoboCOP allows us to solve the collision avoidance. The trajectory of $\dot{p}_y$ shows the slipping contact sequence between the pusher and the slider. Pusher maintains a sticking contact when $\dot{p}_y = 0$.

particular, the initial state of the slider in both these examples was set to $\mathbf{x}_{\texttt{init}} = (0, 0, 0)$ and the goal state for the two conditions was specified as $\mathbf{x}_g = (0.5, 0.5, 0)$ and $(-0.1, -0.1, 3\pi/2)$. We add the obstacles next to the goal state so that PyRoboCOP has to find completely different solution compared to the case when there are no obstacles. The initial point of contact between the pusher and the slider is $p_y = 0$. The optimal pushing sequence to reach the goal states for the slider are shown in Figures 11a and 11c. To provide more insight about the solution, we also provide the plot of the input sequences in Figures 11b and 11d. The slipping contact sequence between the slider and the pusher is seen in the plot of $\dot{p}_y$. Sticking contact occurs when $\dot{p}_y = 0$. We show that the proposed solver can optimize for the desired sequence of contact modes in order to reach the target state. For the example in Figure 11a, the objective is a function of target state and control inputs. For the example in Figure 11c, the Mayer objective function is used.

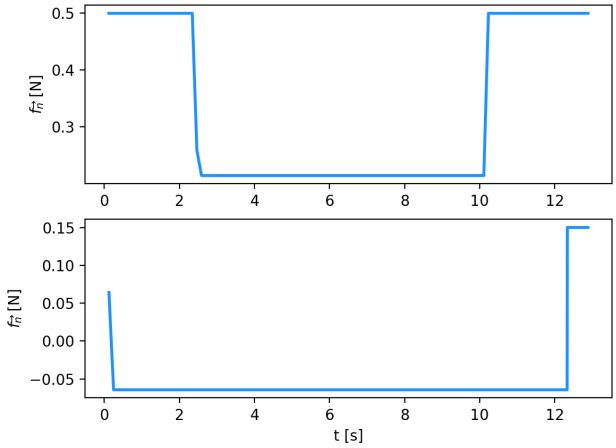
E. Optimization with Mode Enumeration

We show our approach of optimization over fixed mode sequences using the quasi-static pushing model which was presented in Section IV-A while considering sticking contact at the 4 faces of the slider (see Figures 2 and 3). In particular, we use the dynamics model and the problem described in [42] to show solutions obtained by PyRoboCOP in the case where the mode sequence is pre-specified. Note that this can be easily extended to the case where one can search for the mode-sequence using the approach discussed in [42]. Thus, we do not discuss mode sequence search here.

The contact model in this case can be obtained from the model described in Section IV-A, Eq 5 with $\dot{p}_y = 0$. Thus we only consider sticking contact between the pusher and the slider. The modes appear based on which face the pusher contacts with the slider, and thus we have four different modes that could be used during any interaction (see Figure 2 for the four possible modes for the system). It is noted that this is different from the modes considered in the previous sections for pushing, where modes change between sticking and slipping.



(a) Optimal pushing sequence computed by PyRoboCOP.



(b) Optimal control inputs computed by PyRoboCOP.

Fig. 12: Optimal pushing sequence and control inputs obtained by optimizing mode sequence.

For a given mode, state-space of the pusher-slider system is then 3 dimensional while the input is only 2 dimensional. We use our formulation presented in Section IV-C to solve for the optimization problem with pre-specified mode sequence. The optimization process ensures continuity of dynamics and selection of final time for each mode in a trajectory. The initial state of the slider is $\mathbf{x}_{\texttt{init}} = (0, 0, 0)$ and the goal state of the slider is $\mathbf{x}_{\mathbf{g}} = (0, 0, \pi)$. The two modes we use for this example are pushing from the left face followed by pushing from the top face of the slider. The trajectory obtained by PyRoboCOP is shown in Figure 12a. The inputs used in different modes is shown in Figure 12b. The objective function used is the Mayer objective function, i.e., the minimum-time problem. The time spent in mode 1 is 12.36 seconds and in mode 2 is 0.56 seconds.

F. Trajectory Optimization with a Machine Learning Model

We illustrate the usage of PyRoboCOP to control a complex dynamical system such as the circular maze represented

Fig. 13: The circular maze environment for which we show trajectories using the learned Gaussian process models.

in Figure 13. The goal in this system is to tip and tilt the maze in order to move a marble from an outer ring into the inner-most ring. The movement of the maze is actuated by two servomotors. The forward dynamics of the marble moving in the maze are learned using Gaussian Process Regression, as described in our previous work [52]. The model is assumed to be a black box system and ADOL-C is used to compute the derivatives. PyRoboCOP computes the control sequence and the marble's trajectory to reach one of the gates at each of the 4 rings. Figure 14 shows the computed optimal trajectory of

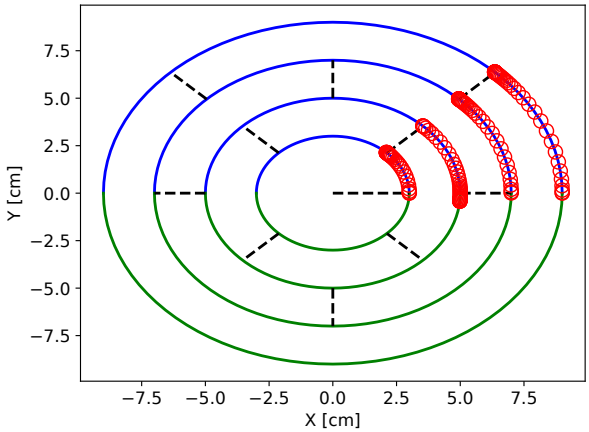


Fig. 14: The red circles represent the marble's optimal trajectory computed by PyRoboCOP for each of the 4 rings of the maze.

the marble in each of the rings, illustrated on a schematic of the maze.

The optimal control sequence in each of the ring is shown in Figure 15.

Remark. *The choice of using ADOL-C gives the opportunity of having any machine learning model written in standard Python and the computation of the derivatives comes automatic together with the non-zero sparsity structure without requiring any insight of the model itself. However, this does not preclude the option that if the derivatives are available*

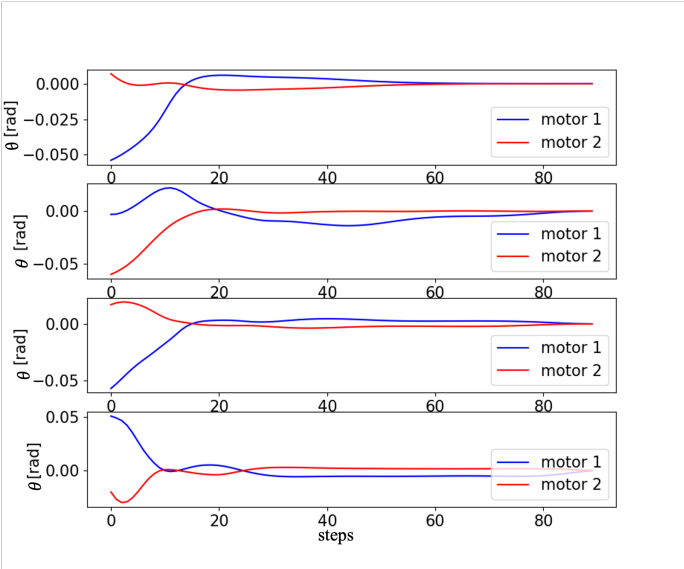


Fig. 15: Optimal control sequence for the maze system in each of the 4 rings.

from other toolboxes e.g., pytorch these derivatives can also be provided to the solver instead than the ones coming from ADOL-C .

G. System Identification For Complementarity Systems

The system identification problem for systems with complementarity constraints is a particular case of the optimization problem (3) and therefore can be solved in PyRoboCOP. The objective is to identify the physical parameters of a system given a set of collected data. As a case of study, we consider the cart-pole with softwalls depicted in Figure 16. The interactions of the pole with the soft walls is modeled as complementarity constrains. The dynamical equations and more details on the system can be found in [53].

We formalize the parameter estimation problem as MPCC (3) where the cost function is the normalized Root Mean Square Error (nRMSE) between the observed trajectory and the trajectory obtained from the estimation procedure. The parameters we aim to identify are the mass of the pole, m_p, and the spring constants of the two walls k_1 and k_2. In PyRoboCOP these parameters are implemented as time independent parameters p. We validated the method with a Monte Carlo simulation on 4 different sets of parameters $p^i = [m_p^i, k_1^i, k_2^i]$ with $i = \{1, \ldots, 4\}$ sampled independently from a uniform distribution, each of which was tested with different levels of independent Gaussian noise added to the trajectories collected. The trajectories are generated with an input sequence that is computed as a sum of sinusoids. Each MC simulation has 50 random realization of the noise. The results are shown in Figure 17 where on the x-axis we have the cart-pole system defined with one of the parameter set p^i and the standard deviation of the noise for each system is in order $[0.0001, 0.001, 0.01, 0.05]$.

We can observe how PyRoboCOP is able to identify both the parameters in the dynamics equations as well as in the complementarity constraints with the lower levels of noise. As seen in the figure, we observe that with higher amounts of noise, the estimation method starts diverging.

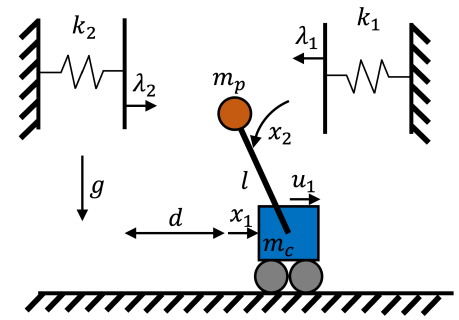


Fig. 16: A schematic of cartpole with softwall system.

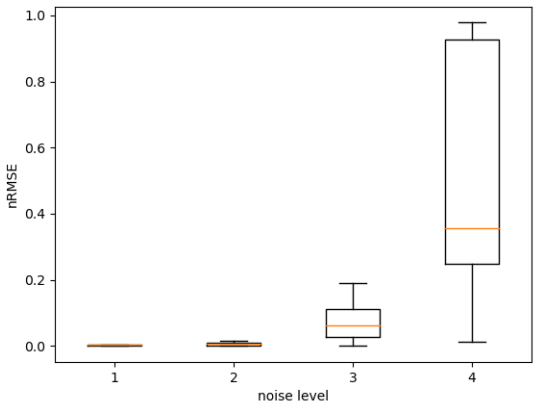


Fig. 17: Distributions of the estimation errors for 4 different cart-pole with softwalls with increasing noise level.

H. Comparison with CasADi and Pyomo

The purpose of this section is to compare PyRoboCOP with some state-of-the-art open-source software for optimization and control, namely CasADi and Pyomo . These two software define their own syntax to model complex nonlinear optimization problems. In particular, CasADi uses a specific symbolic language that allows a fast automatic differentiation and translation to C-code, while Pyomo offers automatic differentation via the AMPL Solver Library. In PyRoboCOP we provide to the user both an interface to specify the model and constraints derivatives manually and an automatic differentiation capability by using ADOL-C which computes the derivatives in C-code and provides access to the sparsity pattern which can be used during optimization. Furthermore, neither CasADi nor Pyomo offer specific constructors to handle complementarity constraints which is one of the major focuses in PyRoboCOP.

We compared the three packages on three different benchmark dynamical systems with increasing state dimension: an inverted pendulum, an acrobot and a quadrotor. The purpose is to see if PyRoboCOP converges to the same solutions in comparable time. For fair comparison the models, the

constraints and the initial conditions are identical in all three software and we used IPOPT as the solver in all the experiments. We solve the OCP for these systems 5 times using each software and record the mean and standard deviation for solution time. Results are shown in Table I.

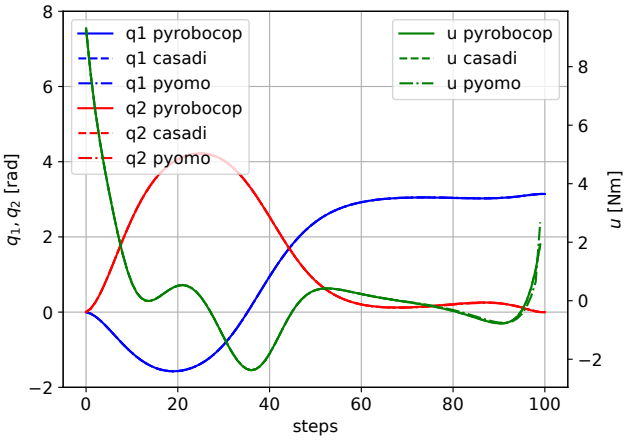


Fig. 18: Optimal trajectory computed by the three optimization packages.

In the table we can first notice that in each experiment the three software converge to a solution with a similar cost function implying convergence to similar optimal solution. An example of this is shown for the acrobot system in Figure 18, where both the optimal state trajectory and the optimal control sequence computed by the three software are plotted, and there is an almost exact overlap. Similar plots have been obtained for the other systems but are not shown for sake of brevity. Second, we can notice that the problem size, which is the total number of variables in the optimization problem is higher in PyRoboCOP w.r.t. the other software. We have chosen a formulation of the transcription that depends only on the number of collocation points n_c and independent of the choice of the roots. With the choice of `legendre` roots in the collocation we observe that the number of variables in Pyomo is larger than that in PyRoboCOP which remains invariant to choice of roots. The analysis of the difference is orthogonal to the main point of this section. It can also be observed that the number of iterations are different in the three optimization packages. This can be attributed to the difference in the problem size for the underlying NLP, which can also be seen in Table I.

We can conclude that all the three software packages are more or less equivalent in solving the considered systems. Based on our experiments, it seems that CasADi tends to outperform the other two in terms of computational time. The main advantages of PyRoboCOP over the other packages are in offering convenient in-built methods to (i) model complementarity constraints and (ii) automatically formulate the collision avoidance constraints for all pairs of user-specified objects using the novel obstacle avoidance formulation.

VII. Concluding Remarks

This paper presented PyRoboCOP which is a Python-based optimization package for model-based control of robotic systems. This package has been developed with the motivation to allow Python-based control of contact-rich systems operating in constrained environments in the presence of other obstacles. We showed that PyRoboCOP can be used to solve trajectory optimization problems of a number of dynamical systems in different configurations such as with contact and collision avoidance constraints. We demonstrated two practical scenarios of planar pushing and belt-drive unit assembly where one needs to consider the collision avoidance as well as contact constraints. A description of the functions that a potential user needs to implement in order to solve their control or optimization problem has been provided. The software has been benchmarked against two other SOTA optimization packages to verify the solution quality and timing obtained by PyRoboCOP.

A. *Strengths of* PyRoboCOP

PyRoboCOP is a model-based trajectory optimization, control and estimation package for systems with non-linear and non-smooth dynamics. In particular, PyRoboCOP can handle systems with contact as well as collision constraints with a novel complementarity formulation. PyRoboCOP also allows automatic differentiation by using ADOL-C. To the best of our knowledge, PyRoboCOP is the only Python-based, open-source software that allows handling of contact & collision constraints and automatic differentiation for control and optimization. Unlike most of the competing optimization solvers which are available in Python, PyRoboCOP allows users to provide dynamics information in Python through a simple script using Numpy data structures [54]. We show that we achieve similar computation times as achieved by other SOTA optimization toolboxes like CasADi and Pyomo. Note that these solvers do not provide adaptive relaxations for complementarity constraints as in PyRoboCOP. Another advantage is that PyRoboCOP allows interfacing to MPCC solver using standard Numpy data structures instead of using any special-purpose data structures designed for PyRoboCOP. This makes PyRoboCOP easier to use when compared to other packages like CasADi and Pyomo. PyRoboCOP also simplifies solution of collision avoidance problems by requiring users to only provide the bounding polytope for each obstacle. A potential user does not need to specify the constraints arising from collision avoidance – this is handled by PyRoboCOP internally. This also reduces the risk of modeling errors made by users when defining these constraints which can be very hard to track down.

B. *Limitations of* PyRoboCOP *and Future Work*

We would also like to highlight some of the limitations of PyRoboCOP. Since PyRoboCOP uses IPOPT as the solver for the resulting MPCC problems, it borrows limitations of IPOPT. In particular, one of the main limitations is that PyRoboCOP can find only local solutions. Furthermore,

System	State Dim	IPOPT Time	Func Eval Time	Problem size	Cost Func	Iterations
Pendulum PyRoboCOP	2	0.112± 0.003	0.101 ± 0.003	1048	19.89	29
Pendulum CasADi	2	0.058 ± 0.001	0.016 ± 0.0008	750	19.89	21
Pendulum Pyomo	2	0.146 ± 0.008	0.008 ± 0.0008	755	20.13	31
Acrobot PyRoboCOP	4	2.282 ± 0.05	1.85 ± 0.019	1296	62.724	349
Acrobot CasADi	4	1.175 ±0.023	0.706 ± 0.008	900	62.52	355
Acrobot Pyomo	4	2.374 ± 0.039	0.652 ± 0.021	909	62.76	265
Quadrotor PyRoboCOP	12	0.871 ± 0.016	0.786 ± 0.013	7988	156.01	21
Quadrotor CasADi	12	0.353 ± 0.002	0.050 ± 0.001	5600	156.01	9
Quadrotor Pyomo	12	1.190 ± 0.038	0.057 ± 0.001	5628	155.64	26

TABLE I: Comparison between PyRoboCOP, CasADi and Pyomo on three non-linear systems of different dimensions. In all cases PyRoboCOP achieves comparable performance with both CasADi and Pyomo.

it might require good initialization to find even the local solutions. Furthermore, it can not detect infeasibility of the underlying optimization problem provided by the user. Another possible limitation is given by interfacing PyRoboCOP with ADOL-C. While, as described above, this is one of the strengths of PyRoboCOP it also carries some limitations as we still have to rely on an external code to do the automatic differentiation while other software like CasADi have built-in source code transformation into C and can handle the differentiation internally with faster performance. In the future we will explore other open source options.

For collision avoidance problems, providing an initial iterate that is collision-free and feasible with respect to the dynamics is important for improved convergence of the optimization solver. Such a trajectory can be obtained using RRT [55] and integration of such techniques with PyRoboCOP will greatly enhance the performance on such problems [56]. We propose to address this in a future work.

While we have shown that PyRoboCOP can handle a broad range of robotic control problems, the current software has been developed under the premise that the dynamics is provided to PyRoboCOP as DAEs, which requires expert knowledge for system specification. We identify this is a limitation as this restricts the developmental usage of PyRoboCOP. In the future, we would work towards integrating PyRoboCOP with a physics engine for easy specification of dynamics to provide solution to complex manipulation tasks with real-time feedback [57], [58], [59]. We will also extend the MPCC formulation presented in the current paper to allow finite horizon model predictive control (MPC) for hybrid systems in future research to allow real-time control of contact-rich systems [60].

We are also interested in extending the functionality of PyRoboCOP to specify uncertain parameters in contact-rich systems. Our recent work [59], [61] has already demonstrated that the framework in PyRoboCOP is conducive to handling such uncertainty. In these papers, we had to explicitly resolve the uncertainty by sampling the uncertain parameters. We are interested in providing users with a convenient method for specifying the the uncertainty in the systems so that the sampling of the uncertain parameters and the resulting formulation can be handled by PyRoboCOP and relieve the user of the burden of having to perform the sampling, reformulation of the problem. The sampling and reformulation will happen in a manner akin to the transcription of the continuous-time DAE to a finite-dimensional problem by discretization.

References

[1] A. Griewank, D. Juedes, and J. Utke, "Algorithm 755: Adol-c: A package for the automatic differentiation of algorithms written in c/c++," *ACM Trans. Math. Softw.*, vol. 22, no. 2, p. 131–167, June 1996.

[2] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, p. 25–57, 2006.

[3] M. T. Mason, "Toward robotic manipulation," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 1–28, 2018.

[4] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[5] K. Ota, D. K. Jha, D. Romeres, J. van Baar, K. A. Smith, T. Semitsu, T. Oiki, A. Sullivan, D. Nikovski, and J. B. Tenenbaum, "Data-efficient learning for complex and real-time physical problem solving using augmented simulation," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 4241–4248, 2021.

[6] L. Biegler, *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Society for Industrial and Applied Mathematics, 2010.

[7] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, Second Edition*, 2nd ed. Society for Industrial and Applied Mathematics, 2010. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898718577

[8] L. Biegler, "An overview of simultaneous strategies for dynamic optimization," *Chemical Engineering and Processing: Process Intensification*, vol. 46, pp. 1043–1053, 2007.

[9] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. D. Prete, "Optimization-based control for dynamic legged robots," 2022.

[10] A. U. Raghunathan, D. K. Jha, and D. Romeres, "Pyrobocop: Python-based robotic control & optimization package for manipulation," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 985–991.

[11] Z. Manchester and S. Kuindersma, "Variational contact-implicit trajectory optimization," in *Robotics Research*. Springer, 2020, pp. 985–1000.

[12] A. Patel, S. L. Shield, S. Kazi, A. M. Johnson, and L. T. Biegler, "Contact-implicit trajectory optimization using orthogonal collocation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2242–2249, 2019.

[13] T. Erez and E. Todorov, "Trajectory optimization for domains with contacts using inverse dynamics," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4914–4919.

[14] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, 2012, pp. 137–144.

[15] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–8, 2012.

[16] K. Yunt and C. Glocker, "Trajectory optimization of mechanical hybrid systems using sumt," in *9th IEEE International Workshop on Advanced Motion Control*, 2005, p. 665–671.

[17] ——, "A combined continuation and penalty method for the determination of optimal hybrid mechanical trajectories," in *IUTAM Symposium on Dynamics and Control of Nonlinear Systems with Uncertainty*. Netherlands: Springer, 2007, p. 187–196.

[18] K. Yunt, "An augmented lagrangian-based shooting method for the optimal trajectory generation of switching lagrangian systems," *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications & Algorithms*, vol. 18, p. 615–645, 2011.

[19] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[20] X. Zhang, A. Liniger, A. Sakai, and F. Borrelli, "Autonomous parking using optimization-based collision avoidance," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 4327–4332.

[21] P. Kolaric, D. K. Jha, A. U. Raghunathan, F. L. Lewis, M. Benosman, D. Romeres, and D. Nikovski, "Local policy optimization for trajectory-centric reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5094–5100.

[22] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, "Modeling and optimization with optimica and jmodelica.org—languages and tools for solving large-scale dynamic optimization problems," *Computers & Chemical Engineering*, vol. 34, no. 11, pp. 1737–1749, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S009813540900283X

[23] B. Houska, H. Ferreau, and M. Diehl, "Acado toolkit – an open source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, p. 298–312, 2011.

[24] S. Leyffer and C. Kirches, "Taco - a toolkit for ampl control optimization," *Mathematical Programming Computation*, p. 1–39, 2013.

[25] B. Nicholson, J. Siirola, J. Watson, Z. V.M., and L. Biegler, "pyomo.dae: a modeling and automatic discretization framework for optimization with differential and algebraic equations," *Mathematical Programming Computation*, vol. 10, p. 187–223, 2018.

[26] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[27] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[28] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 2536–2542.

[29] D. M. Murray and S. J. Yakowitz, "Constrained differential dynamic programming and its application to multireservoir control," *Water Resources Research*, vol. 15, no. 5, pp. 1017–1027, 1979.

[30] A. N. Alexandros Filotheou and D. V. Dimarogonas, "Robust decentralised navigation of multi-agent systems with collision avoidance and connectivity maintenance using model predictive controllers," *International Journal of Control*, vol. 93, no. 6, pp. 1470–1484, 2020. [Online]. Available: https://doi.org/10.1080/00207179.2018.1514129

[31] A. Nikou, C. Verginis, S. Heshmati-alamdari, and D. V. Dimarogonas, "A nonlinear model predictive control scheme for cooperative manipulation with singularity and collision avoidance," in *2017 25th Mediterranean Conference on Control and Automation (MED)*, 2017, pp. 707–712.

[32] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*, 1st ed. Talor & Francis, 1975.

[33] Z.-Q. Luo, J.-S. Pang, and D. Ralph, *Mathematical programs with equilibrium constraints*. Cambridge University Press, 1996.

[34] A. Raghunathan and L. Biegler, "An interior point method for mathematical programs with complementarity constraints (mpccs)," *SIAM J. Optimization*, vol. 15, no. 3, pp. 720–750, 2005.

[35] A. De Miguel, M. Friedlander, F. Nogales, and S. Scholtes, "A two-sided relaxation scheme for mathematical programs with equilibrium constraints," *SIAM J Optimization*, vol. 16, no. 2, p. 587—609, 2005.

[36] S. Leyffer, G. López-Calva, and J. Nocedal, "Interior methods for mathematical programs with complementarity constraints," *SIAM J. Optimization*, no. 1, p. 52–77, 2006.

[37] M. Anitescu, P. Tseng, and S. Wright, "Elastic-mode algorithms for mathematical programs with equilibrium constraints: global convergence and stationarity properties," *Mathematical Programming*, vol. 110, no. 2, pp. 337–371, 2007.

[38] J. S. Pang and D. E. Stewart, "Differential variational inequalities," *Mathematical Programming*, vol. 113, no. 2, pp. 345–424, 2008.

[39] A. Patel, S. Shield, S. Kazi, A. Johnson, and L. Biegler, "Contact-implicit trajectory optimization using orthogonal collocation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2242–2249, 2019.

[40] A. Patel, S. L. Shield, S. Kazi, A. M. Johnson, and L. T. Biegler, "Contact-implicit trajectory optimization using orthogonal collocation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, 2019.

[41] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 533–556, 1996.

[42] N. Doshi, F. R. Hogan, and A. Rodriguez, "Hybrid differential dynamic programming for planar manipulation primitives," 2020.

[43] W. Zhang, S. Seto, and D. K. Jha, "CAZSL: Zero-shot regression for pushing models by generalizing through context," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. [Online]. Available: https://arxiv.org/abs/2003.11696.pdf

[44] N. Chavan-Dafle, R. Holladay, and A. Rodriguez, "Planar in-hand manipulation via motion cones," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 163–182, 2020.

[45] F. R. Hogan, E. R. Grau, and A. Rodriguez, "Reactive planar manipulation with convex hybrid mpc," 2018.

[46] M. Bauza, F. R. Hogan, and A. Rodriguez, "A data-efficient approach to precise and controlled pushing," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 336–345. [Online]. Available: http://proceedings.mlr.press/v87/bauza18a.html

[47] S. Goyal, A. Ruina, and J. Papadopoulos, "Planar sliding with dry friction part 1. limit surface and moment function," *Wear*, vol. 143, no. 2, pp. 307–330, 1991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0043164891901043

[48] E. Gilbert and D. Johnson, "Distance functions and their application to robot path planning in the presence of obstacles," *IEEE Journal on Robotics and Automation*, vol. 1, no. 1, pp. 21–30, 1985.

[49] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[50] F. von Drigalski, C. Schlette, M. Rudorfer, N. Correll, J. Triyonoputro, W. Wan, T. Tsuji, and T. Watanabe, "Robots assembling machines: Learning from the world robot summit 2018 assembly challenge," 2019.

[51] S. Jin, D. Romeres, A. Raghunathan, D. K. Jha, and M. Tomizuka, "Trajectory optimization for manipulation of deformable objects: Assembly of belt drive units," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021. [Online]. Available: https://arxiv.org/abs/2106.00898

[52] D. Romeres, D. K. Jha, A. DallaLibera, B. Yerazunis, and D. Nikovski, "Semiparametrical gaussian processes learning of forward dynamical models for navigating in a circular maze," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3195–3202.

[53] A. Aydinoglu, P. Sieg, V. M. Preciado, and M. Posa, "Stabilization of complementarity systems via contact-aware controllers," *IEEE Transactions on Robotics*, 2021.

[54] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sept. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[55] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[56] M. Zhang, D. K. Jha, A. U. Raghunathan, and K. Hauser, "Simultaneous trajectory optimization and contact selection for multi-modal manipulation planning," *arXiv preprint arXiv:2306.06465*, 2023.

[57] S. Dong, D. Jha, D. Romeres, S. Kim, D. Nikovski, and A. Rodriguez, "Tactile-RL for insertion: Generalization to objects of unknown geometry," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021. [Online]. Available: https://arxiv.org/pdf/2104.01167.pdf

[58] Y. Shirai, D. K. Jha, A. U. Raghunathan, and D. Romeres, "Robust pivoting: Exploiting frictional stability using bilevel optimization," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 992–998.

[59] Y. Shirai, D. K. Jha, A. Raghunathan, and D. Romeres, "Chance-constrained optimization in contact-rich systems for robust manipulation," *arXiv preprint arXiv:2203.02616*, 2022.

[60] Y. Shirai, D. K. Jha, A. U. Raghunathan, and D. Hong, "Tactile tool manipulation," *arXiv preprint arXiv:2301.06698*, 2023.

[61] Y. Shirai, D. K. Jha, and A. U. Raghunathan, "Covariance steering for uncertain contact-rich systems," 2023.

Arvind U Raghunathan received his Bachelor of Technology degree (summa cum laude) from Indian Institute of Technology, Madras and Ph.D. degree Carnegie Mellon University both in Chemical Engineering in 1999 and 2004, respectively. He is currently the Senior Principal Research Scientist and Senior Team Leader of the Optimization & Intelligent Robotics Team at Mitsubishi Electric Research Laboratories, Cambridge, MA, USA. His research interests include development of optimization algorithms and their applications to electric power operations, control of robotic systems, and operations of transportation systems. Arvind's research has been recognized with the 2022 IEEE Control Systems Society Roberto Tempo Best CDC Paper Award.

Devesh K. Jha (Senior Member, IEEE) is currently a Principal Research Scientist at Mitsubishi Electric Research Laboratories (MERL) in Cambridge, MA, USA. He received PhD in Mechanical Engineering from Penn State in Decemeber 2016. He received M.S. degrees in Mechanical Engineering and Mathematics also from Penn State. His research interests are in the areas of Machine Learning, Robotics and Deep Learning. He is a recipient of several best paper awards including the Kalman Best Paper Award 2019 from the American Society of Mechanical Engineers (ASME) Dynamic Systems and Control Division. He is a senior member of IEEE and an associate editor of IEEE Robotics and Automation Letters (RA-L).

Diego Romeres (Senior Member, IEEE) received his M.Sc. degree (summa cum laude) in control engineering and the Ph.D. degree in information engineering from the University of Padua, Padua, Italy, in 2012 and 2017, respectively. He is currently a Principal Research Scientist and Team Leader of the Intelligent Robotics Team at Mitsubishi Electric Research Laboratories, Cambridge, MA, USA. He held visiting research positions at TU Darmstadt, Darmstadt, Germany, and at ETH, Zurich, Switzerland. His research interests include robotics, artificial intelligence, machine learning, reinforcement learning, bayesian optimization and system identification theory. He is serving as an associate editor of IEEE International Conference on Robotics and Automation (ICRA) and International Conference on Intelligent Robots and Systems (IROS).