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

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

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

TR2022-057 June 03, 2022

Abstract

PYROBOCOP is a Python-based package for con-trol, optimization and estimation of robotic systems described by nonlinear Differential Algebraic Equations (DAEs). In par-ticular, the package can handle systems with contacts that are described by complementarity constraints and provides a gen-eral framework for specifying obstacle avoidance constraints. The package performs direct transcription of the DAEs into a set of nonlinear equations by performing orthogonal collocation on finite elements. PYROBOCOP provides automatic reformu-lation of the complementarity constraints that are tractable to 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 evaluate PYROBOCOP on several manipulation problems for control and estimation.

IEEE International Conference on Robotics and Automation (ICRA) 2022

© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

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

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

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

***Abstract*— PyRoboCOP is a Python-based package for control, optimization and estimation of robotic systems described by nonlinear Differential Algebraic Equations (DAEs). In particular, the package can handle systems with contacts that are described by complementarity constraints and provides a general framework for specifying obstacle avoidance constraints. The package performs direct transcription of the DAEs into a set of nonlinear equations by performing orthogonal collocation on finite elements. PyRoboCOP provides automatic reformulation of the complementarity constraints that are tractable to 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 evaluate PyRoboCOP on several manipulation problems for control and estimation.**

I. Introduction

Most manipulation applications are characterized by presence of constrained environments while dealing with challenging underlying phenomena like unilateral contacts, frictional contacts, impact and deformation [3]. With this understanding, we present a python-based robotic control and optimization package (called PyRoboCOP) that allows solution to a large class of mathematical programs with nonlinear and complementarity 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.

Contact-rich robotic manipulation tasks could be modeled as complementarity systems. Obtaining a feasible, let alone an optimal trajectory, can be challenging for such systems. An effective integration of the high-level trajectory planning in configuration 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 the 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. The formulation in PyRoboCOP 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:

- Contact modeling by complementarity constraints
- Obstacle avoidance modeling by complementarity constraints
- Automatic differentiation for sparse derivatives
- Support for minimum time problems
- Support for optimization over fixed mode sequence problems with unknown sequence time horizons
- Support for parameter estimation in linear complementarity systems.

The features described above should convince the reader that PyRoboCOP addresses lots of important optimization problems for manipulation 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 manipulation systems. Codes and instructions for installing and using PyRoboCOP could be found in [4] under a research only license.

Contributions. The main contributions of the paper are:

1) We present a python-based package for optimization and control of robotic systems with contact and collision constraints.
2) We evaluate our proposed package, PyRoboCOP , over a range of different manipulation systems for control and estimation.

For the sake of space, we do not report the formulation of collision avoidance with complementarity constraints. The interest reader can refer to the arXiv version of the work [5].

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 [6], [7], [8], [9], [10], [11], [12], [13]. 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 [14], minimum distance (e.g., in collision avoidance) [15], or in some cases robustness constraints [16].

Some of the existing open-source software for dynamic optimization are Optimica [17], ACADO Toolkit [18], TACO [19], pyomo.dae [20], Drake [21] and CasADi [22]. 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 convenient interfaces for handling complementarity constraints and obstacle-avoidance which are key requirements

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

in robotic applications. More recently, some packages have been proposed to perform contact-rich tasks in robotics [23]. However, the solver proposed in [23] uses DDP-based [24] techniques which suffer from sub-optimality and difficulty in constraint satisfaction. Compared to most of the other techniques in open literature, PYROBOCOP implements different 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 [11], [12], [13]. The authors pose contact dynamics as a measure differential inclusion and employ an augmented Lagrangian to solve the resulting complementarity constrained optimization problem. [14] 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 package that can handle constraints arising due to frictional contact interaction and collision avoidance.

III. PROBLEM DESCRIPTION

PYROBOCOP solves the dynamic optimization problem

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

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

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

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

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. The function $\phi:\mathbb{R}^{n_x+n_p}\to\mathbb{R}$ represents Mayer-type objective function [25] 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 $\widehat{x}$. 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) using the Implicit Euler time-stepping scheme. 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 NLP that results is

$$\min \sum_{i=1}^{N_e} h_i c(x_i,y_i,u_{i-1})+\phi(x_{N_e},p) \tag{2a}$$

$$\text{s.t. } f(\dot{x}_i,x_i,y_i,u_{i-1})=0,\; x_0=\widehat{x} \tag{2b}$$

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

$$\underline{x}\le x_i\le\overline{x}, \underline{y}\le y_i\le\overline{y}, \underline{u}\le u_i\le\overline{u} \tag{2d}$$

$$x_{i+1}=x_i+h_i\dot{x}_i \tag{2e}$$

where the decision variables in (2) are x_i, $\dot{x}_i$, y_i and u_i. The subscript i approximates the value of the corresponding variable at time t_i. The constraints (2b)-(2c) are imposed for $i\in\mathcal{N}_e$. The constraint in (2e) models Implicit Euler time-stepping scheme and is imposed for $i\in\mathcal{N}_e\setminus\{N_e\}$. If complementarity constraints are present then (2) 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 [26]. Hence, solution of MPCCs has warranted careful handling of the complementarity constraints when used in Interior Point Methods for NLP (IPM-NLP) using relaxation [27], [28] or penalty formulations [29]. In the case of active set methods, the robust solution of MPCC relies on special mechanism such as the elastic mode [30].

PYROBOCOP implements two possible relaxation schemes for complementarity constraints

$$\alpha_l([y_i]_{\sigma_{l,1}}-\nu_{l,1})([y_i]_{\sigma_{l,2}}-\nu_{l,2})\le\delta\,\forall\, l\in\mathcal{L} \tag{3a}$$

$$\sum_{l\in\mathcal{L}}\alpha_l([y_i]_{\sigma_{l,1}}-\nu_{l,1})([y_i]_{\sigma_{l,2}}-\nu_{l,2})\le\delta \tag{3b}$$

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 (2d) are satisfied. The first approach relaxes each complementarity constraint by a positive parameter δ [27] while the second approach imposes the relaxation on the summation of all the complementarity constraints over a finite element i [30]. 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 [27], [28].

IV. SOFTWARE DESCRIPTION

Figure 1 provides a high-level summary of the flow of control in PYROBOCOP. Detailed descriptions on the

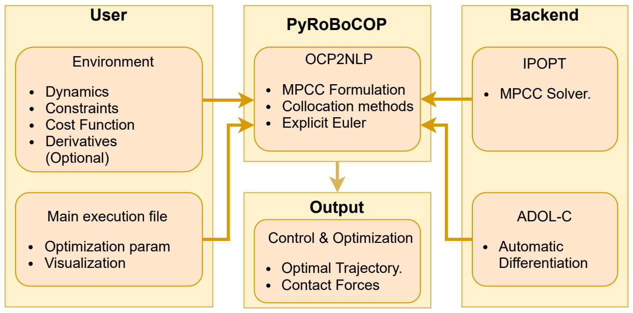


Fig. 1: Workflow in PyRoboCOP. The dynamics provided by the user to create a MPCC which is then optimized using IPOPT and the gradients are evaluated using automatic differentiation via ADOL-C.

software API and classes are available in the Software Description in [4]. A user provided class specifies the dynamic optimization problem (1). This is also briefly described in Figure 1. 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 1).

- `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 to 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 $n_O > 0$.

PyRoboCOP is interfaced with ADOL-C [1] to compute derivatives (see the Backend block in Figure 1). 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 (2a), (ii) constraints including the DAE (2b) and a reformulation of (2c), and (iii) the Hessian of the Lagrangian of the NLP (2). 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 (2a), (2b) 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 1, PyRoboCOP uses IPOPT as the optimization solver. The choice of the solver agrees with our desire of having an open source software package, for this reason IPOPT is chosen over other competitors like SNOPT [31] and filterSQP [32].

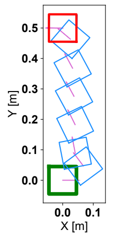
The user has flexibility in specifying how the complementarity constraints are solved. The choices are: (i) (3a) with δ fixed, (ii) (3b) with δ fixed, (iii) (3a) with δ set equal to the interior point barrier parameter, (iv) (3b) with δ set equal to the interior point barrier parameter, and (v) the objective function is appended with complementarity terms as $\sum_{i=1}^{N_e} \sum_{l\in\mathcal{L}} \alpha_l([y_i]_{\sigma_{l,1}} - \nu_{l,1})([y_i]_{\sigma_{l,2}} - \nu_{l,2})$. 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. As empirical guidelines, we recommend options (iii) and (iv) since they enforce the complementarity constraints in a gradual manner as the algorithm converges to a solution.

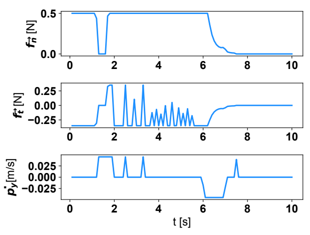
V. Numerical Results

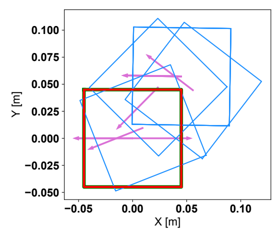
In this section, we test PyRoboCOP in several robotic simulations providing solutions to trajectory optimization problems including several systems with complementarity constraints. To foster reproducibility, the source code for each of the following examples is available in [4].

A. Planar Pushing

In this section, we show some results for planar pushing. We briefly describe the dynamic model here .For more de-


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


(b) Optimal Controls


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

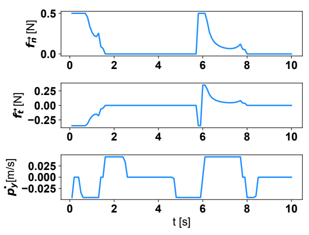

(d) Optimal Controls

Fig. 2: 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 only few frames of the trajectory in Fig. 2c.

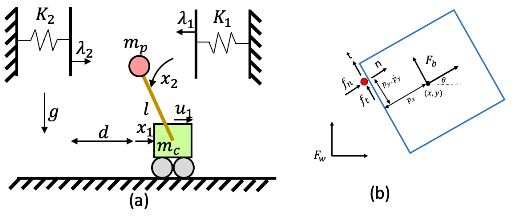


Fig. 3: A schematic of cartpole with softwalls and planar pusher-slider system we study in this paper.

tailed description of the pushing model, readers are referred to [33] and [34]. A schematic for a pusher-slider system is shown in Figure 3. The pusher interacts with the slider by exerting forces in the normal and tangential direction denoted by $f_{\vec{n}}$, $f_{\vec{t}}$ (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 [35] 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 $\mathbf{w} = [f_{\vec{n}}, f_{\vec{t}}, \tau]$ could be written as $\mathbf{w} = \mathbf{J}^T(\vec{n} f_{\vec{n}} + \vec{t} f_{\vec{t}})$. For the contact configuration shown in Figure 3, the normal and tangential unit vectors are given by $\vec{n} = [1 \quad 0]^T$ and $\vec{t} = [0 \quad 1]^T$.

The equations of motion of the pusher-slider system are

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{Rt} & \dot{p}_y \end{bmatrix}^\top \tag{4}$$

where $\mathbf{R}$ is the rotation matrix. 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 $\mathbf{u} = [f_{\vec{n}} \quad f_{\vec{t}} \quad \dot{p}_y]^T$. The elements of the input vector must follow the laws of coulomb friction which can be expressed as complementarity conditions as follows:

$$\begin{aligned} 0 \le \dot{p}_{y+} \perp (\mu_p f_{\vec{n}}(t) - f_{\vec{t}}(t)) \ge 0 \\ 0 \le \dot{p}_{y-} \perp (\mu_p f_{\vec{n}}(t) + f_{\vec{t}}(t)) \ge 0 \end{aligned} \tag{5}$$

where $\dot{p}_y = \dot{p}_{y+} - \dot{p}_{y-}$ and the μ_p is the coefficient of friction between pusher and slider. The complementarity conditions in Eq. (5) 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 and $\dot{p}_y$ is non-zero only at the boundary of friction-cone.

Two pushing trajectories with different goal configurations from the same initial state are shown in Figure 2. 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.

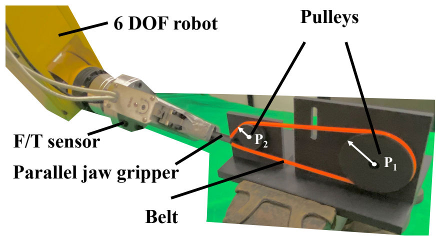
B. Assembly of Belt Drive Unit

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[1] [36]. The real world system is represented in Figure 4a 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.

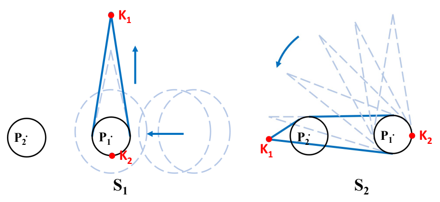
The full manipulation task has been divided into two subtasks as shown in Figure 4b. The goals of the first and second subtask are to wrap the belt around the first and second pulley, respectively. We formulate two trajectory optimization problems one for each of the two subtasks as a MPCC of the form in (1).

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

[1]`https://worldrobotsummit.org/en/about/`



(a) Real Setup of the Belt Drive Unit system.



(b) Visualization of two subtasks decomposition, S_1 and S_2. P_1 and P_2 are the pulleys.

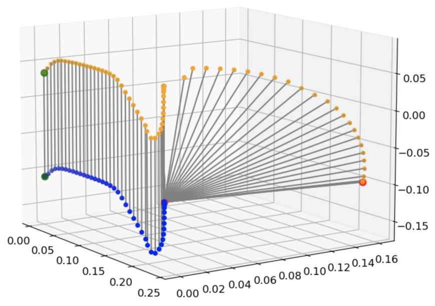
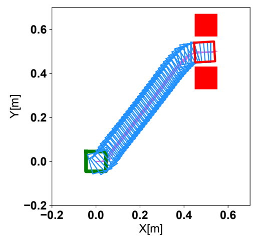
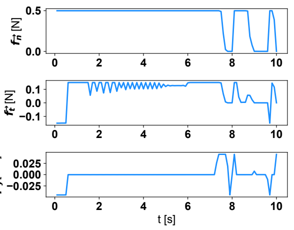
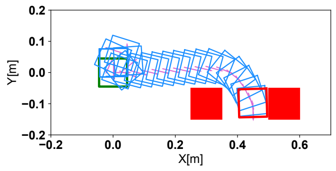

(c) Optimal trajectory to assemble the belt.

Fig. 4: BDU system is shown in (a). In (b) the blue lines represent the belt gripped by a robot at keypoint K_1, and K_2 is the lower keypoint. In figure (c) 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 belt approaches the first pulley (not shown), then there is a downwards movement to hook the pulley with the lower keypoint during S_1. K_1 is now 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 to conclude S_2.


(a) Pushing sequence with $\mathbf{x}_{\texttt{init}}$ $(0, 0, 0)$ and $\mathbf{x_g} = (0.5, 0.5, 0)$


(b) Optimal Controls obtained for Example 5a


(c) Pushing sequence $\mathbf{x}_{\texttt{init}}$ $(0, 0, 0)$ and $\mathbf{x_g} = (0.45, -0.1, \frac{3\pi}{2})$

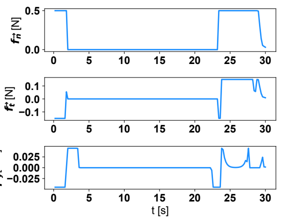

(d) Optimal Controls obtained for Example 5c.

Fig. 5: 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 $\dot{p}_y = 0$.

found in our previous paper [37]. In Figure 4c 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 [37] for further details.

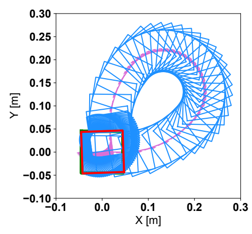
C. Planar Pushing With Obstacles

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 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 5a and 5c. To provide more insight about the solution, we also provide the plot of the input sequences in Figures 5b and 5d. 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 5a, the objective is a function of target state and control inputs. For the example in Figure 5c, the Mayer objective function is used.

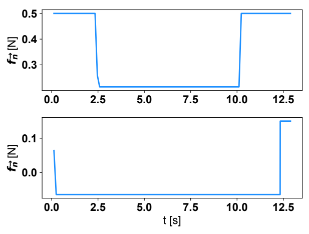
D. Optimization with Mode Enumeration

We show our optimization approach over fixed mode sequences using the quasi-static pushing model presented in Section V-A while considering sticking contact at the 4 faces of the slider (see Figure 3). In particular, we use the dynamics model and the problem described in [38] to show solutions obtained by PyRoboCOP in the case the mode sequence is pre-specified and the resulting problem is feasible.

The contact model in this case can be obtained from the model described in Section V-A, Eq 4 with $\dot{p}_y = 0$. 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. For a given mode, state-space of the pusher-slider system is then 3 dimensional while



(a) Optimal pushing sequence.



(b) Optimal control inputs.

Fig. 6: Optimal pushing sequence and control inputs obtained by optimizing mode sequence with PyRoboCOP.

the input is only 2 dimensional. 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 6a. The inputs used in different modes is shown in Figure 6b. The objective function used is the Mayer objective function, i.e., the minimum-time problem. The time spent in mode 1 is 12.36[s] and in mode 2 is 0.56[s].

E. System Identification For Complementarity Systems

The system identification problem for systems with complementarity constraints is a particular case of the optimization problem (2) 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 3. 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 [39].

We formalize the parameter estimation problem as MPCC (2) 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 $i = \{1, \dots, 4\}$ 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 7 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

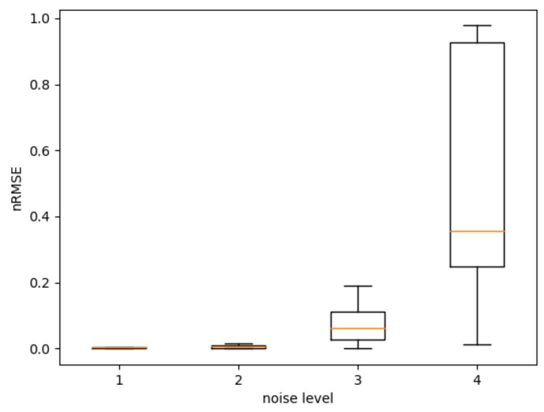


Fig. 7: Distributions of the estimation errors for 4 different cart-pole with softwalls with increasing noise level.

figure, we observe that with higher amounts of noise, the estimation method starts diverging.

VI. Concluding Remarks

This paper presented PyRoboCOP, a Python-based optimization package for model-based control of robotic systems. We showed that PyRoboCOP can be used to solve trajectory optimization problems of a number of dynamical systems in different configurations in presence of contact and collision avoidance 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. More details are available in the software package [4].

Strengths of PyRoboCOP 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, refer to the software description in [4].

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 only local solutions can be found. Furthermore, good initialization to find even the local solutions might be required. Finally, infeasibility of the underlying optimization problem provided by the user cannot be detected. 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 have to rely on an external code to do automatic differentiation, while other software like CasADi have built-in source code transformation into C and can handle the differentiation internally with faster performance.

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, Jun. 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] A. U. Raghunathan, D. K. Jha, and D. Romeres. (2021) Pyrobocop. [Online]. Available: https://www.merl.com/research/license/PyRoboCOP

[5] ——, "Pyrobocop : Python-based robotic control & optimization package for manipulation and collision avoidance," 2021.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] ——, "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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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

[18] 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.

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[32] R. Fletcher and S. Leyffer, "Nonlinear programming without a penalty function," *Mathematical programming*, vol. 91, no. 2, pp. 239–269, 2002.

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

[34] 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

[35] 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

[36] 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.

[37] S. Jin, D. Romeres, A. Ragunathan, 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

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

[39] A. Aydinoglu, V. M. Preciado, and M. Posa, "Stabilization of complementarity systems via contact-aware controllers," *arXiv preprint arXiv:2008.02104*, 2020.